```
;                 bytes 0-5 = Multicast Address.
;                 bytes 6-7 = Entry used(Non zero if used).
;
; On Entry:       EAX     N/A
;                 EBX     N/A
;                 ECX     # of Entries in Table( 0 if empty )
;                 EDX     N/A
;                 EBP     @ Adapter Data Space
;                 ESI     @ Multicast Table
;                 EDI     N/A
;
;                 Note:   Interrupts are in any state.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Destroyed
;                 EDI     Destroyed
;
;                 Flags:
;
;                 Note:   Interrupts preserved.
;
; Remarks:        This routine is called by the ethernet media module.
;                 It can be called at process or interrupt time.
;
; See Also:       ETHERTSM\EtherTSMAddMulticastAddress
;                 ETHERTSM\EtherTSMDeleteMulticastAddress
;                 ETHERTSM\EtherTSMUpdateMulticast
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
DriverMulticastChange   proc
;
;*************************************************************\
;                                                              *
; First reset Multicast Address Registers.                     *
;                                                              *
;*************************************************************/
;
        ret

DriverMulticastChange   endp
        subttl  -- DriverPromiscuousChange --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverPromiscuousChange, DPC/API/PROMISCU )
;
; Name:           DriverPromiscuousChange
;
; Description:    This routine will enable/disable the Promiscuous Mode.
;
; On Entry:       EAX     N/A
;                 EBX     N/A
;                 ECX     0 to disable the Promiscuous mode
;                 EDX     N/A
;                 EBP     @ Adapter Data Space
;                 ESI     @ Multicast Table
;                 EDI     N/A
;
```

```
;                 Note:   Interrupts are in any state.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Destroyed
;                 EDI     Destroyed
;
;                 Flags:
;
;                 Note:   Interrupts preserved.
;
; Remarks:        This routine is called by the ethernet media module.
;                 It can be called at process or interrupt time.
;
; See Also:       ETHERTSM\EtherTSMPromiscuousChange
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
DriverPromiscuousChange proc

        ret

DriverPromiscuousChange endp
        subttl  -- CalculateDriftDelta --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( CalculateDriftDelta, DPC/API/CALCDD )
;
; Name:           CalculateDriftDelta
;
; Description:    Acquisition State Routine.
;
; On Entry:       EAX     N/A
;                 EBX     Frame Data Space
;                 ECX     N/A
;                 EDX     N/A
;                 EBP     Adapter Data Space
;                 ESI     N/A
;                 EDI     N/A
;
;                 Note:   Interrupts are in any state.
;
; On Return:      EAX     Destroyed
;                 EBX     Preserved
;                 ECX     Destroyed
;                 EDX     Destroyed
;                 EBP     Preserved
;                 ESI     Preserved
;                 EDI     Preserved
;
;                 Flags:
;
;                 Note:   Interrupts preserved.
;
; Remarks:        This routine is called by InitState.
;                 It can be called at process or interrupt time.
;
; See Also:
;
```

```
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  CalculateDriftDelta
CalculateDriftDelta     proc


        mov     edi, [ebp].Drift
        cmp     edi, NOM_COUNT_TRACK
        jbe     DriftBelowNOM

        lea     eax, [edi - NOM_COUNT_TRACK]
        xor     edx, edx
        mov     ecx, 210
        div     ecx
        mov     [ebp].GLDrift, eax

        mov     ecx, 210
        mul     ecx
        shr     eax, 4
        mov     edi, eax
        mov     eax, [ebp].Drift
        sub     eax, NOM_COUNT_TRACK
        sub     eax, edi

        mov     edi, 0ffffh
        sub     edi, [ebp].GLDrift
        inc     edi
        mov     [ebp].GLDrift, edi

        ret

DriftBelowNOM:
        mov     eax, NOM_COUNT_TRACK
        sub     eax, [ebp].Drift
        xor     edx, edx
        mov     ecx, 210
        div     ecx
        mov     [ebp].GLDrift, eax

        mov     ecx, 210
        mul     ecx
        shr     eax, 4
        mov     edi, NOM_COUNT_TRACK
        sub     edi, [ebp].Drift
        sub     edi, eax

        mov     eax, 0ffffh
        sub     eax, edi
        inc     eax
        ret

CalculateDriftDelta     endp
        subttl  -- Step --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( Step, DPC/API/STEP )
;
; Name: /       Step
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
```

```
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by InitState.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  Step
Step    proc

        mov     eax, [ebp].NextStepCount
        inc     eax
        xor     edx, edx
        mov     ecx, 4
        div     ecx
        mov     [ebp].NextStepCount, edx

        mov     eax, [ebp].SearchLoc
        cmp     [ebp].SearchLocFound, FALSE
        je      DontUseNextStep

        or      edx, edx
        je      StepSetGLOffset
        cmp     edx, 2
        je      StepSetGLOffset

        inc     eax
        cmp     edx, 1
        je      StepDivideBy3
        inc     eax
StepDivideBy3:
        xor     edx, edx
        mov     ecx, 3
        div     ecx
        mov     eax, edx

StepSetGLOffset:
        mov     eax, SearchTbl[eax * 4]
        mov     [ebp].GLOffset, eax
        jmp     StepCalcRx

DontUseNextStep:
        mov     ecx, SearchTbl[eax * 4]
        mov     [ebp].GLOffset, ecx

        inc     eax
        xor     edx, edx
        mov     ecx, 3
        div     ecx
        mov     [ebp].GLOffset, edx

StepCalcRx:

        mov     [ebp].Drift, 0
        call    CalculateRxFreq
        mov     [ebp].Drift, NOM_COUNT_TRACK
        ret

Step    endp
        subttl  -- InitState --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( InitState, DPC/API/INITSTA )
;
; Name:         InitState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  InitState
InitState       proc

        cmp     [ebp].TrackingMode, TRUE
        jne     InitStateNotTracking

        cmp     DebugMask, 0
        je      InitStateNoMsg
```

```
        mov     eax, offset InitStateTrackMsg
        cmp     eax, LastDebugMessage
        je      InitStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
InitStateNoMsg:

        call    CalculateDriftDelta
        add     eax, NOM_COUNT_REACQ

        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al
        shr     al, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOGateCountHighAddr
        mov     eax, [ebp].ReacqGateCount
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        xor     eax, eax
        out     dx, ax

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     InitStateSetMode
        and     al, NOT MODE_MASK
        jmp     InitStateCheckRate

InitStateSetMode:
        or      al, MODE_MASK
InitStateCheckRate:
        out     dx, al

        mov     edx, [ebp].IOSpareIOControlAddr
        mov     al, 0ah
        cmp     [ebp].ViterbiOnly, 2
        je      InitStateSetRate
        mov     al, 0bh
        cmp     [ebp].ViterbiOnly, 1
        je      InitStateSetRate
        mov     al, 0fh
InitStateSetRate:
        out     dx, al
        mov     [ebp].NextState, ENABLE_BTR
        jmp     InitStateCheckPointing

InitStateNotTracking:

        cmp     DebugMask, 0
        je      InitStateNNoMsg
        mov     eax, offset InitStateNoTrackMsg
        cmp     eax, LastDebugMessage
        je      InitStateNNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
InitStateNNoMsg:

        mov     edx, [ebp].IOBitDetControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOSpareIOControlAddr
        mov     al, 0ah
        cmp     [ebp].ViterbiOnly, 2
        je      InitStateNTSetRate
        mov     al, 0bh
        cmp     [ebp].ViterbiOnly, 1
        je      InitStateNTSetRate
        mov     al, 0fh
InitStateNTSetRate:
        out     dx, al

        mov     edx, [ebp].IODaAdOffsetControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        mov     eax, [ebp].ModulationScheme
        or      eax, SWEEP_DIR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 8ah
        out     dx, al

        mov     edx, [ebp].IOGateCountHighAddr
        mov     eax, [ebp].ReacqGateCount
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].SqfDeltaCount
        out     dx, al

        mov     eax, [ebp].NomCountSearch
        mov     [ebp].TuneCount, eax
        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al
        shr     eax, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
```

```
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOAgcFirControlAddr
        mov     al, 10 OR AGC_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
        out     dx, al

        mov     edx, [ebp].IOCrlkThrLowAddr
        mov     al, 60h
        out     dx, al

        mov     edx, [ebp].IOCthAddr
        mov     al, 0e0h
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        mov     al, SENA_MASK
        cmp     [ebp].ModulationScheme, BPSK
        jne     InitStateSetSena
        or      al, DEPUNC_BYPASS_MASK
InitStateSetSena:
        out     dx, al

        mov     edx, [ebp].IOCrlkControlAddr
        mov     al, 16 OR CRLK_GAIN_OFFSET OR CRLK_DET_PWR_OFFSET
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     InitStateResetBtr

        or      al, RESET_BTR_ACC_MASK
        and     al, NOT MODE_MASK
        jmp     InitStateClearBtr
InitStateResetBtr:
        or      al, MODE_MASK OR RESET_BTR_ACC_MASK
InitStateClearBtr:
        out     dx, al

        in      al, dx
        and     al, NOT RESET_BTR_ACC_MASK
        out     dx, al

        call    Step

        mov     [ebp].NextState, ACQ_PD

InitStateCheckPointing:

        mov     [ebp].RateCount, 0
        mov     [ebp].PointingFlag, TRUE
        cmp     [ebp].DemodCommand, POINTING_MODE
        je      InitStateExit
        mov     [ebp].PointingFlag, FALSE
InitStateExit:
        mov     [ebp].CurrentState, SYNTH_PRGM
        mov     [ebp].SignalQuality, 0
        mov     [ebp].DemodCommand, BUSY_MODE
        mov     [ebp].DemodStatus, UNLOCKED
        mov     [ebp].FecStatus, UNLOCKED
        ret

InitState       endp
        subttl  -- ProgTuner --
        page
;
;*************************************************************************\
;
; BEGIN_MANUAL_ENTRY( ProgTuner, DPC/API/PROGTUN )
;
; Name:         ProgTuner
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by Tune.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*************************************************************************/
;
        public  ProgTuner
ProgTuner       proc
;
; EAX = data
; ECX = len
;
        dec     ecx
        mov     edx, 1
        shl     edx, cl
        mov     ecx, edx
        mov     esi, eax                ; ESI = Data
```

```
ProgTunerLoop:
        jecxz   ProgTunerExit

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        test    esi, ecx
        je      ProgTunerClear

        or      al, SDATA_MASK
        and     al, NOT SCLK_MASK
        out     dx, al
        jmp     ProgTunerDelay

ProgTunerClear:
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

ProgTunerDelay:
        shr     ecx, 1

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SCLK_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        jmp     ProgTunerLoop

ProgTunerExit:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SCLK_MASK
        out     dx, al

        ret

ProgTuner       endp
        subttl  -- Tune --
        page
;
;*************************************************************************\
;
; BEGIN_MANUAL_ENTRY( Tune, DPC/API/TUNE )
;
; Name:         Tune
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by SynthPrgmState.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  Tune
Tune    proc

        cmp     [ebp].TunerTypeFound, SHARP
        je      TuneSharpPan
        cmp     [ebp].TunerTypeFound, PANASONIC
        je      TuneSharpPan
        cmp     [ebp].TunerTypeFound, SHARP_CUSTOM
        je      TuneSharpCustom
        ret

TuneSharpPan:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

        cmp     [ebp].TrackingMode, 0
        jne     TuneSetNA

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, 2ch
        cmp     [ebp].TunerTypeFound, SHARP
        je      TuneProgTuner
        mov     eax, 0ech
TuneProgTuner:
        mov     ecx, 8
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx
```

```
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, 28h OR 2000h
        mov     ecx, 16
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

TuneSetNA:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, [ebp].ChannelNumber
        add     eax, [ebp].GLDrift
        xor     edx, edx
        mov     ecx, SYNTH_RATIO
        div     ecx
        mov     edi, edx
        or      eax, 3000h

        mov     ecx, 16
        call    ProgTuner

        mov     eax, edi
        mov     ecx, 8
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        ret

TuneSharpCustom:
        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT (SCLK_MASK OR SDATA_MASK)
        out     dx, al

        mov     eax, 50h OR 8001h
        mov     ecx, 16
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        mov     eax, [ebp].ChannelNumber
        add     eax, [ebp].GLDrift
        xor     edx, edx
        mov     ecx, SYNTH_RATIO
        div     ecx
        mov     edi, edx

        mov     ecx, 11
        call    ProgTuner

        mov     eax, edi
        shl     eax, 1
        mov     ecx, 9
        call    ProgTuner

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, SENA_MASK
        out     dx, al

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        in      al, dx

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT SENA_MASK
        out     dx, al

        ret

Tune    endp
        subttl  -- SynthPrgmState --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( SynthPrgmState, DPC/API/SYNTHPS )
;
; Name:         SynthPrgmState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
```

```
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  SynthPrgmState
SynthPrgmState  proc

        cmp     DebugMask, 0
        je      SynthPrgmStateNoMsg
        mov     eax, offset SynthPrgmMsg
        cmp     eax, LastDebugMessage
        je      SynthPrgmStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
SynthPrgmStateNoMsg:
        call    Tune

        mov     [ebp].TrackingMode, 0
        cmp     [ebp].NextState, ACQ_PD
        jne     SynthPrgmClearT2

        mov     [ebp].MaxSqf, 0
        mov     [ebp].SqfAvg, 0
        mov     [ebp].SqfWait, 0
        mov     eax, [ebp].SqfCheckPoints
        mov     [ebp].MaxCount, eax
        mov     [ebp].T2Count, 60
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

SynthPrgmClearT2:
        mov     [ebp].T2Count, 0
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

SynthPrgmState  endp
        subttl  -- AcqPDDelayState --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( AcqPDDelayState, DPC/API/ACQPDDS )
;
; Name:         AcqPDDelayState
;
; Description:  Acquisition State Routine.
;
; On Entry      EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*************************************************************************/
;
        public  AcqPDDelayState
AcqPDDelayState proc

        cmp     DebugMask, 0
        je      AcqPDDelayStateNoMsg
        mov     eax, offset AcqPDDelayMsg
        cmp     eax, LastDebugMessage
        je      AcqPDDelayStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
AcqPDDelayStateNoMsg:
        cmp     [ebp].T2Count, 0
        jne     AcqPDDelayExit

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 87h
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        and     al, NOT SQF_PEAK_EN_MASK
        out     dx, al

        mov     eax, [ebp].SqfWait
        mov     [ebp].T2Count, eax

        mov     eax, [ebp].NextState
        mov     [ebp].CurrentState, eax

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        or      al, SQF_PEAK_EN_MASK
        out     dx, al
```

```
AcqPDDelayExit:
        ret

AcqPDDelayState endp
        subttl  -- AcqPDState --
        page
;
;*************************************************************************\
;
; BEGIN_MANUAL_ENTRY( AcqPDState, DPC/API/ACQPDS )
;
; Name:         AcqPDState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*************************************************************************/
;
        public  AcqPDState
AcqPDState      proc

        cmp     DebugMask, 0
        je      AcqPDStateNoMsg
        mov     eax, offset AcqPDMsg
        cmp     eax, LastDebugMessage
        je      AcqPDStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
AcqPDStateNoMsg:
        cmp     [ebp].T2Count, 0
        jne     AcqPDExit

        xor     eax, eax
        mov     edx, [ebp].IOMaxSqfAddr
        in      al, dx
        add     [ebp].SqfAvg, eax

        cmp     eax, [ebp].MaxSqf
        jbe     AcqPDDecMaxCount

        mov     [ebp].MaxSqf, eax
        mov     eax, [ebp].TuneCount
        mov     [ebp].BestTuneCount, eax

AcqPDDecMaxCount:
        dec     [ebp].MaxCount
        jne     AcqPDMaxCountNotZero

        mov     edx, [ebp].IOSweepRateAddr
        mov     al, 8ah
        out     dx, al

        mov     edx, [ebp].IOCountNomLowAddr
        mov     eax, [ebp].BestTuneCount
        out     dx, al
        shr     eax, 8
        mov     edx, [ebp].IOCountNomHighAddr
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].SqfDeltaCount
        shl     eax, 1
        out     dx, al

        mov     eax, [ebp].SqfAvg
        mov     ecx, [ebp].SqfCheckPoints
        xor     edx, edx
        div     ecx
        add     eax, 2
        mov     [ebp].SqfAvg, eax

        mov     [ebp].T2Count, 40
        mov     [ebp].NextState, ENABLE_BTR
        mov     [ebp].CurrentState, ACQ_PD_DELAY
AcqPDExit:
        ret

AcqPDMaxCountNotZero:
        mov     eax, [ebp].TuneCount
        add     eax, [ebp].SqfCheckStepSize
        mov     [ebp].TuneCount, eax

        mov     edx, [ebp].IOCountNomLowAddr
        out     dx, al

        mov     edx, [ebp].IOCountNomHighAddr
        shr     eax, 8
        out     dx, al

        mov     [ebp].T2Count, 20
        mov     [ebp].CurrentState, ACQ_PD_DELAY
        ret

AcqPDState      endp
        subttl  -- EnableBTRState --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( EnableBTRState, DPC/API/ENBTRST )
```

```
;
; Name:             EnableBTRState
;
; Description:      Acquisition State Routine.
;
; On Entry:         EAX     N/A
;                   EBX     Frame Data Space
;                   ECX     N/A
;                   EDX     N/A
;                   EBP     Adapter Data Space
;                   ESI     N/A
;                   EDI     N/A
;
;                   Note:   Interrupts are in any state.
;
; On Return:        EAX     Destroyed
;                   EBX     Preserved
;                   ECX     Destroyed
;                   EDX     Destroyed
;                   EBP     Preserved
;                   ESI     Preserved
;                   EDI     Preserved
;
;                   Flags:
;
;                   Note:   Interrupts preserved.
;
; Remarks:          This routine is called by DriverCallBack.
;                   It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  EnableBTRState
EnableBTRState  proc

        cmp     DebugMask, 0
        je      EnableBTRStateNoMsg
        mov     eax, offset EnableBTRMsg
        cmp     eax, LastDebugMessage
        je      EnableBTRStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
EnableBTRStateNoMsg:
        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        or      al, BTR_ERR_ENA_MASK
        out     dx, al

        mov     [ebp].CurrentState, START_SEARCH_FOR_FEC
        ret

EnableBTRState  endp
        subttl  -- StartSearchForFECState --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( StartSearchForFECState, DPC/API/SRCHFEC )
;
```

```
; Name:          StartSearchForFECState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  StartSearchForFECState
StartSearchForFECState  proc

        cmp     DebugMask, 0
        je      StartSearchForFECStateNoMsg
        mov     eax, offset StartSearchForFECMsg
        cmp     eax, LastDebugMessage
        je      StartSearchForFECStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
StartSearchForFECStateNoMsg:
        cmp     [ebp].PointingFlag, 0
        je      SearchFECNotPointing

        mov     [ebp].CurrentState, POINTING_ACQ
        mov     eax, [ebp].SqfAvg
        add     eax, 2
        mov     [ebp].MaxSqf, eax
        jmp     SearchFECSetMax

SearchFECNotPointing:

        mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
        mov     eax, [ebp].SqfAvg
        add     eax, 6
        mov     [ebp].MaxSqf, eax

SearchFECSetMax:
```

```
        mov     edx, [ebp].IOCthAddr
        out     dx, al

        mov     edx, [ebp].IOAfcControlAddr
        in      al, dx
        and     al, NOT SWP_ENA_MASK
        out     dx, al

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].T1Count, 300

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT RESET_FEC_ACQ_MASK
        out     dx, al

        ret

StartSearchForFECState  endp
        subttl  -- CheckforFECLockState --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( CheckforFECLockState, DPC/API/CHKFEC )
;
; Name:          CheckforFECLockState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  CheckforFECLockState
```

```
CheckforFECLockState    proc

        cmp     DebugMask, 0
        je      CheckForFECLockStateNoMsg
        mov     eax, offset CheckforFECLockStateMsg
        cmp     eax, LastDebugMessage
        je      CheckForFECLockStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
CheckForFECLockStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        and     al, FEC_LOCK_MASK
        je      CheckFECNotLocked

        mov     [ebp].DemodStatus, LOCKED
        mov     [ebp].FecStatus, LOCKED

        mov     edx, [ebp].IOGateCountHighAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].ReacqDeltaCount
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out     dx, al

        in      al, dx
        or      al, 5
        out     dx, al

        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 500
        mov     [ebp].T2Count, 100
        mov     [ebp].CurrentState, TRACKING
        ret

CheckFECNotLocked:

        cmp     [ebp].T1Count, 0
        jne     CheckFECHaveT1Count

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, CRL_LOCK_MASK
        jne     CheckFECSetOtherMode

        mov     [ebp].CurrentState, INIT
        ret

CheckFECSetOtherMode:
        mov     [ebp].CurrentState, SET_OTHER_MODE
CheckFECExit:
        ret

CheckFECHaveT1Count:
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
```

```
        test    al, CRL_LOCK_MASK
        jne     CheckFECExit
        mov     eax, [ebp].MaxSqf
        cmp     eax, [ebp].SqfAvg
        jbe     CheckFECExit
        sub     eax, 2
        mov     [ebp].MaxSqf, eax
        mov     edx, [ebp].IOCthAddr
        out     dx, al
        ret

CheckforFECLockState    endp
        subttl  -- SetOtherModeState --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( SetOtherModeState, DPC/API/SETOTHER )
;
; Name:         SetOtherModeState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  SetOtherModeState
SetOtherModeState       proc

        cmp     DebugMask, 0
        je      SetOtherModeStateNoMsg
        mov     eax, offset SetOtherModeStateMsg
        cmp     eax, LastDebugMessage
        je      SetOtherModeStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
SetOtherModeStateNoMsg:

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        cmp     [ebp].ViterbiMode, LOWRATE
        jne     SetOtherModeToLow

        or      al, MODE_MASK
        out     dx, al

        mov     [ebp].ViterbiMode, HIGHRATE
        jmp     SetOtherModeIncRate

SetOtherModeToLow:
        and     al, NOT MODE_MASK
        out     dx, al

        mov     [ebp].ViterbiMode, LOWRATE

SetOtherModeIncRate:
        inc     [ebp].RateCount

        cmp     [ebp].RateCount, 1
        jg      SetOtherModeExit

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        or      al, RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].T1Count, 180

        mov     edx, [ebp].IOSynthSerControlAddr
        in      al, dx
        and     al, NOT RESET_FEC_ACQ_MASK
        out     dx, al

        mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
        ret

SetOtherModeExit:
        mov     [ebp].CurrentState, INIT
        ret

SetOtherModeState       endp
        subttl  -- ReadWord --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( ReadWord, DPC/API/READWORD )
;
; Name:         ReadWord
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
```

```
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by TrackingState,
;               PointingAcquisitionState and PointingTrackingState
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  ReadWord
ReadWord        proc

        push    ebx

        xor     ebx, ebx
        mov     edx, edi
        in      al, dx
        mov     bh, al

        mov     ecx, 4
ReadWordLoop:
        mov     edx, esi
        in      al, dx
        mov     bl, al

        mov     edx, edi
        in      al, dx
        cmp     al, bh
        je      ReadWordExit

        mov     bh, al
        dec     ecx
        jne     ReadWordLoop

ReadWordExit:
        mov     eax, ebx

        pop     ebx
        ret

ReadWord        endp
        subttl  -- TrackingState --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( TrackingState, DPC/API/TRCKST )
;
; Name:         TrackingState
;
; Description:  Acquisition State Routine.
;
```

```
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  TrackingState
TrackingState   proc

        cmp     DebugMask, 0
        je      TrackingStateNoMsg
        mov     eax, offset TrackingStateMsg
        cmp     eax, LastDebugMessage
        je      TrackingStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
TrackingStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, FEC_LOCK_MASK
        jne     TrackingStateReadQuality

        in      al, dx
        test    al, CRL_LOCK_MASK
        je      TrackingStateZero
        cmp     [ebp].T2Count, 0
        jne     TrackingStateT1

TrackingStateZero:
        mov     [ebp].TrackingMode, 0
        mov     [ebp].CurrentState, INIT
        ret

TrackingStateT1:
        mov     [ebp].T1Count, 1000
        ret
```

```
TrackingStateReadQuality:
        xor     eax, eax
        mov     edx, [ebp].IORelSqfAddr
        in      al, dx
        mov     [ebp].SignalQuality, eax

        cmp     DebugMask, 0
        je      SignalStrengthNoMsg
        cmp     LastSignalStrength, 0
        jne     SignalStrengthNoMsg

        mov     LastSignalStrength, 1
        cmp     eax, 200
        jb      SignalStrengthNone
        sub     eax, 200
        shl     eax, 1
        add     eax, 60
        jmp     SignalStrengthPrint
SignalStrengthNone:
        xor     eax, eax
SignalStrengthPrint:
        push    eax
        push    offset SignalStrengthMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (3 * 4)]

SignalStrengthNoMsg:

        cmp     [ebp].T1Count, 0
        jne     TrackingStateExit

        mov     [ebp].T1Count, 1000
        mov     [ebp].TrackingMode, 1

        mov     edi, [ebp].IOTuningHighAddr
        mov     esi, [ebp].IOTuningLowAddr
        call    ReadWord
        mov     [ebp].Drift, eax

        mov     ecx, 2
        cmp     eax, NOM_COUNT_TRACK + OFFSET_THRESHOLD
        ja      TrackingStateLocFound
        mov     ecx, 0
        cmp     eax, NOM_COUNT_TRACK - OFFSET_THRESHOLD
        jb      TrackingStateLocFound
        mov     ecx, 1
TrackingStateLocFound:
        mov     [ebp].SearchLoc, ecx
        mov     [ebp].SearchLocFound, TRUE
TrackingStateExit:
        ret

TrackingState   endp
        subttl  -- PointingAcquisitionState --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( PointingAcquisitionState, DPC/API/PTACQST )
;
; Name:         PointingAcquisitionState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        public  PointingAcquisitionState
PointingAcquisitionState        proc

        cmp     DebugMask, 0
        je      PointingAcquisitionStateNoMsg
        mov     eax, offset PointingAcqStateMsg
        cmp     eax, LastDebugMessage
        je      PointingAcquisitionStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
PointingAcquisitionStateNoMsg:

        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        test    al, SWEEPING_MASK
        je      PointingNotSweeping

        mov     esi, [ebp].IOTuningLowAddr
        mov     edi, [ebp].IOTuningHighAddr
        call    ReadWord
        shl     eax, 4
        mov     [ebp].Drift, eax

        mov     [ebp].DemodStatus, LOCKED

        xor     eax, eax
        mov     edx, [ebp].IOGateCountHighAddr
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in      al, dx
        and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out     dx, al
```

```
        in      al, dx
        or      al, 5
        out     dx, al

        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 1000
        mov     [ebp].SearchLocFound, FALSE
        mov     [ebp].CurrentState, POINTING_TRACKING
        ret

PointingNotSweeping:
        mov     eax, [ebp].MaxSqf
        sub     eax, 2
        mov     [ebp].MaxSqf, eax
        mov     edx, [ebp].IOCthAddr
        out     dx, al
        cmp     [ebp].T1Count, 0
        jne     PointingAcqExit

        mov     [ebp].CurrentState, INIT

PointingAcqExit:
        ret

PointingAcquisitionState        endp
        subttl  -- PointingTrackingState --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( PointingTrackingState, DPC/API/PTTRKST )
;
; Name:         PointingTrackingState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks·      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
```

```
;***********************************************************************/
;
        public  PointingTrackingState
PointingTrackingState   proc

        cmp     DebugMask, 0
        je      PointingTrackingStateNoMsg
        mov     eax, offset PointingTrackStateMsg
        cmp     eax, LastDebugMessage
        je      PointingTrackingStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
PointingTrackingStateNoMsg:
        xor     eax, eax
        mov     edx, [ebp].IORelSqfAddr
        in      al, dx
        mov     [ebp].SignalQuality, eax

        cmp     [ebp].T1Count, 0
        je      PointingTrackingExit

        mov     [ebp].T1Count, 1000

        mov     esi, [ebp].IOTuningLowAddr
        mov     edi, [ebp].IOTuningHighAddr
        call    ReadWord

        mov     ecx, [ebp].Drift
        add     ecx, OFFSET_THRESHOLD
        cmp     eax, ecx
        ja      PointingTrackingInit

        mov     ecx, [ebp].Drift
        sub     ecx, OFFSET_THRESHOLD
        cmp     eax, ecx
        jae     PointingTrackingExit

PointingTrackingInit:
        mov     [ebp].CurrentState, INIT

PointingTrackingExit:
        ret

PointingTrackingState   endp
        subttl  -- HaltState --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( HaltState, DPC/API/HALTST )
;
; Name:         HaltState
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
```

```
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  HaltState
HaltState       proc

        cmp     DebugMask, 0
        je      HaltStateNoMsg
        mov     eax, offset HaltStateMsg
        cmp     eax, LastDebugMessage
        je      HaltStateNoMsg
        mov     LastDebugMessage, eax
        push    eax
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
HaltStateNoMsg:
        ret

HaltState       endp
        subttl  -- InitDemod --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( InitDemod, DPC/API/INITDMOD )
;
; Name:         InitDemod
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;***********************************************************************/
;
        public  InitDemod
InitDemod       proc

        mov     [ebp].CurrentState, HALT
        mov     [ebp].RxFreq, 0
        mov     [ebp].ViterbiMode, 0
        mov     [ebp].DemodCommand, HALT_MODE
        mov     [ebp].SearchLoc, 1
        mov     [ebp].Drift, 0
        mov     [ebp].GLOffset, 0
        mov     [ebp].TrackingMode, FALSE


        ;value = read_bits (STATUS_ADDR, TUNER_TYPE_MASK);
        xor     eax, eax
        mov     edx, [ebp].IOStatusAddr
        in      al, dx
        and     al, TUNER_TYPE_MASK
        mov     cl, al

        ;value |= read_bits (UNIT_ID_ADDR, TUNER_TYPE_2_MASK);
        mov     edx, [ebp].IOUnitIDAddr
        in      al, dx
        and     al, TUNER_TYPE_2_MASK
        or      al, cl

        cmp     al, SHARP
        jne     InitDemodPanasonic

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset SharpTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
        pop     eax
        jmp     FillInTunerVars

InitDemodPanasonic:
        cmp     al, PANASONIC
        jne     InitDemodSharpCustom

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset PanasonicTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
```

```
        pop     eax
        jmp     FillInTunerVars

InitDemodSharpCustom:
        cmp     al, SHARP_CUSTOM
        jne     InitDemodExit

        cmp     DebugMask, 0
        je      FillInTunerVars
        push    eax
        push    offset SharpCustomTunerMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (2 * 4)]
        pop     eax

FillInTunerVars:
        mov     [ebp].TunerTypeFound, eax
        mov     [ebp].ReacqGateCount, 0f0h
        mov     [ebp].ReacqDeltaCount, 2
        mov     [ebp].NomCountSearch, NOM_COUNT_REACQ - 75
        mov     [ebp].SqfCheckPoints, 11
        mov     [ebp].SqfCheckStepSize, 15
        mov     [ebp].SqfDeltaCount, 8
InitDemodExit:
        ret

InitDemod       endp
        subttl  -- ApplyDelay --
        page
;
;***********************************************************************\
;
; BEGIN_MANUAL_ENTRY( ApplyDelay, DPC/API/APPLYDEL )
;
; Name:         ApplyDelay
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
```

```
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  ApplyDelay
ApplyDelay      proc

;
; Apply delay to T1 Counter
;
        cmp     [ebp].T1Count, 0
        je      ApplyDelayT2
        cmp     [ebp].T1Count, eax
        jb      ApplyDelayClearT1
        sub     [ebp].T1Count, eax
        jmp     ApplyDelayT2
ApplyDelayClearT1:
        mov     [ebp].T1Count, 0
;
; Apply delay to T2 Counter
;
ApplyDelayT2:
        cmp     [ebp].T2Count, 0
        je      ApplyDelayExit
        cmp     [ebp].T2Count, eax
        jb      ApplyDelayClearT2
        sub     [ebp].T2Count, eax
        jmp     ApplyDelayExit
ApplyDelayClearT2:
        mov     [ebp].T2Count, 0

ApplyDelayExit:
        ret

ApplyDelay      endp
        subttl  -- CalculateRxFreq --
        page
;
;**********************************************************************\
;
; BEGIN_MANUAL_ENTRY( CalculateRxFreq, DPC/API/CALCRXFQ )
;
; Name:         CalculateRxFreq
;
; Description:  Acquisition State Routine.
;
; On Entry:     EAX     N/A
;               EBX     Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
```

```
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by DriverCallBack.
;               It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
;**********************************************************************/
;
        public  CalculateRxFreq
CalculateRxFreq proc

        ;  USHORT_T freq, total, new_total, results;
        sub     esp, 2 * 4

        ; total = [esp + 0]
        ; results = [esp + 4]
        ; freq = esi
        ; new_total = edi

        ;   freq = S.Rx_Freq - FREQ_BASE;
        ;   freq += S.GL_offset;
        mov     esi, [ebp].RxFreq
        sub     esi, FREQ_BASE
        add     esi, [ebp].GLOffset       ; ESI = freq

        ;   total = (freq * 2) / 729;
        mov     eax, esi                  ; EAX = freq
        shl     eax, 1                    ; EAX = freq * 2
        xor     edx, edx
        mov     ecx, 729
        div     ecx                       ; EAX = EAX / 729
        mov     [esp + 0], eax            ; total = EAX

        ;   results = (total * 729) / 2;
        mov     ecx, 729
        mul     ecx                       ; EAX = total * 729
        shr     eax, 1                    ; EAX = EAX / 2
        mov     [esp + 4], eax            ; results = eax

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total = total * 10;
        mov     eax, [esp + 0]
        mov     ecx, 10
        mul     ecx                       ; EAX = total * 10
        mov     edi, eax                  ; new_tottal = EAX

        ;   total = (freq * 20) / 729;
        mov     eax, esi                  ; EAX = freq
        mov     ecx, 20
        mul     ecx                       ; EAX = freq * 20
        xor     edx, edx
        mov     ecx, 729
        div     ecx                       ; EAX = EAX / 729
        mov     [esp + 0], eax            ; total = EAX

        ;   results = (total * 729) / 20;
        mov     ecx, 729
        mul     ecx                       ; EAX = total * 729
        xor     edx, edx
        mov     ecx, 20
        div     ecx                       ; EAX = EAX / 20
        mov     [esp + 4], eax          ; results = EAX

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total += total;
        add     edi, [esp + 0]

        ;   new_total *= 10;
        mov     eax, edi
        mov     ecx, 10
        mul     ecx                     ; EAX = new_total * 10
        mov     edi, eax                ; EDI = EAX

        ;   total = (freq * 200) / 729;
        mov     eax, esi
        mov     ecx, 200
        mul     ecx                     ; EAX = freq * 200
        xor     edx, edx
        mov     ecx, 729
        div     ecx                     ; EAX = EAX / 729
        mov     [esp + 0], eax          ; total = EAX

        ;   results = (total * 729) / 200;
        mov     ecx, 729
        mul     ecx                     ; EAX = total * 729
        xor     edx, edx
        mov     ecx, 200                ; EAX = EAX / 200
        div     ecx
        mov     [esp + 4], eax          ; results = EAX

        ;   freq = freq - results;
        sub     esi, eax

        ;   new_total += total;
        add     edi, [esp + 0]

        ;   if (freq >= 2) new_total++;
        cmp     esi, 2
        jb      CalcGetChannelNumber
        inc     edi                     ; new_total++
CalcGetChannelNumber:

        ;   S.Channel_Number = SYNTH_FIRST_CHANNEL + new_total;
        add     edi, SYNTH_FIRST_CHANNEL
        mov     [ebp].ChannelNumber, edi

        cmp     DebugMask, 0
        je      NoChannelMsg
        push    edi
        push    offset ChannelNumberMsg
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (3 * 4)]
NoChannelMsg:

        add     esp, 2 * 4
        ret

CalculateRxFreq endp
        subttl  -- DriverCallBack --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverCallBack, DPC/API/CALLBACK )
```

```
;
; Name:         DriverCallBack
;
; Description:  This routine will be executed once every second. It will
;               detect if the hardware does not ack a transmission. If the
;               hardware didn't ack then it will be reset, the transmission
;               of that packet will be aborted and the next packet in the
;               queue will be sent if there is one.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;               After this call returns, the MSM will schedule another
;               call back.
;               It is called at interrupt time.
;
; See Also:     MSM\MSMCallBackProcedure
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
        public  DriverCallBack
        align   16
DriverCallBack  proc

        cmp     [ebp].TunerTypeFound, INVALID_TUNER
        jne     DemodInitialized
        call    InitDemod
;
; Check Rx Frequency
;
DemodInitialized:
;;      cmp     [ebp].RxFreq, 1330 * 10
;;      je      CallBackCheckState
;;      mov     [ebp].RxFreq, 1330 * 10

        cmp     [ebp].RxFreq, 0
        jne     CallBackCheckState
        mov     eax, GlobalRxFreq
;
; RxFreq = GlobalRxFreq * 10
;
        mov     ecx, 10
        mul     ecx
        mov     [ebp].PxFreq, eax
        cmp     [ebp].TrackingMode, FALSE
        je      CheckRxFreqSetMode
        call    CalculateRxFreq
CheckRxFreqSetMode:
        mov     [ebp].DemodCommand, ACQUIRE_MODE
;
; Possibly set the CurrentState depending on DemodCommand
;
CallBackCheckState:
        cmp     [ebp].DemodCommand, ACQUIRE_MODE
        je      CallBackSetInit
        cmp     [ebp].DemodCommand, POINTING_MODE
        jne     CallBackCheckHalt
CallBackSetInit:
        mov     [ebp].CurrentState, INIT
        call    CallBackApplyDelay
        jmp     CallBackWatchDog

CallBackCheckHalt:
        cmp     [ebp].DemodCommand, HALT_MODE
        jne     CallBackApplyDelay
        mov     [ebp].CurrentState, HALT
;
; Apply delay
;
CallBackApplyDelay:
        mov     eax, 15
        call    ApplyDelay

        mov     eax, [ebp].CurrentState
        mov     esi, StateTbl[eax * 4]
        call    esi

CallBackWatchDog:

        ret

        mov     eax, [ebp].BufferCount
        cmp     eax, [ebp].WatchBufferCount
        mov     [ebp].WatchBufferCount, eax
        jne     CallBackExit

        call    RefreshMipsStats

        mov     eax, [ebp].MipsZeroAddrFrames              ; LocalMipsStats.zeroAdd
rFrames
        cmp     eax, [ebp].WatchOldRejected
        mov     [ebp].WatchOldRejected, eax
        je      CallBackExit

        call    DriverISR

        mov     edx, [ebp].PicAddress
        in      al, dx
        SLOW
        or      eax, [ebp].PicMask
        out     dx, al
        SLOW
        in      al, dx
        SLOW
        and     eax, [ebp].PicUnMask
        out     dx, al

CallBackExit:
        ret
```

```
DriverCallBack  endp
        public DriverSend
        subttl  -- DriverSend --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverSend, DPC/API/SEND )
;
; Name:         DriverSend
;
; Description:  This routine will transfer the packet described in the
;               TCB to the NIC and initiate the send. TxStartTime and
;               RetryCounter must be set to enable the deadman timer.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     Padded Packet Length
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     @ TCB
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process or interrupt time.
;
; See Also:     ETHERTSM\EtherTSMDriverSend
;               ETHERTSM\MediaSendRaw8023
;               ETHERTSM\MediaSendEthernetII
;               ETHERTSM\MediaSend8022Over8023
;               ETHERTSM\MediaSend8022Snap
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        align   16
DriverSend      proc

        lea     edi, [esi].TCBMediaHeader
        cmp     word ptr [edi+12], 0608h        ; ARP(0x08 0x06)?
        je      DriverSendArp                   ; Jump if it is

        cmp     [ebp].AgentSendRoutine, 0       ; Can we send it yet?
        je      DriverSendExit                                          ; Give i
t back if we can't

        push    ecx
; Padded size
        push    esi
; Address of TCB
        call    [ebp].AgentSendRoutine  ; Give it to Slip Handler
        pop     esi
        pop     ecx

        ret

DriverSendExit:
        inc     [ebp].MSMTxFreeCount            ; Add a send resource
        jmp     EtherTSMFastSendComplete        ; Otherwise service events.

DriverSendArp:
;
; We're going to assume that the entire request is in the first
; fragment. Verify it first.
;
        mov     edi, [esi].TCBFragStrucPtr
        cmp     dword ptr [edi+0], 1            ; One fragment?
        jne     DriverSendExit          ; Jump if not
        cmp     dword ptr [edi+8], 28   ; Entire ARP request(frag length)?
        jb      DriverSendExit          ; Jump if not
;
; Make sure sender and target ip addr are different
;
        mov     edi, [edi+4]                    ; EDI -> ARP request

        mov     eax, [edi+28-14]                ; EAX = senders IP
        cmp     eax, [edi+38-14]                ; Same as target IP?
        je      DriverSendExit          ; Jump out if it is

        push    esi                             ; Save send ECB
        push    edi                             ; Save ARP offset

        mov     esi, 1514                       ; Max ECB size.
        call    MSMAllocateRCB          ; Get an ECB
        pop     edi                             ; EDI -> ARP request again
        or      eax, eax
        jne     DriverSendReturnARP             ; Jump if no ECB.
;
; ESI -> reply ECB
; EDI -> request data
;
        lea     eax, [esi+RPacketEnvelope]      ; EAX -> beginning
        mov     [esi].RPacketOffset, eax        ; Store into ECB
        mov     [esi].RPacketSize, 60   ; ARP reply size
        mov     [esi].RPacketLength, 60 ;  (ethernet min size)

        push    esi                             ; Save reply ECB
        mov     esi, eax                        ; ESI -> reply data
;
; ESI -> reply data
; EDI -> request data
;
;
; Set reply->dest_addr to our node address
;
        mov     eax, dword ptr [ebx].MLIDNodeAddress+0
        mov     dword ptr [esi+0], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     word ptr [esi+4], ax
;
; Set reply->source_addr to (0x06 0x06 0x06 0x06 0x06 0x06)
;
        mov     word ptr [esi+6], 0606h
        mov     dword ptr [esi+8], 06060606h
;
; Set reply->type to (0x08 0x06)
;
```

```
        mov     word ptr [esi+12], 0608h
;
; Set reply hardware type(0x00 0x01), protocol type(0x08 0x00)
;
        mov     dword ptr [esi+14], 00080100h
;
; Set reply hardware size(0x06), protocol size(0x04)
;    and operation(0x00 0x02 for ARP reply)
;
        mov     dword ptr [esi+18], 02000406h
;
; Set reply senders ethernet addr to (0x06 0x06 0x06 0x06 0x06 0x06).
;
        mov     dword ptr [esi+22], 06060606h
        mov     word ptr [esi+26], 0606h
;
; Set reply senders ip addr to the request target ip addr
;
        mov     eax, [edi+38-14]                ; request->target_ip
        mov     [esi+28], eax
;
; Set reply target ethernet addr to our node addr
;
        mov     eax, dword ptr [ebx].MLIDNodeAddress+0
        mov     dword ptr [esi+32], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     word ptr [esi+36], ax
;
; Set reply target ip addr to request senders ip addr
;
        mov     eax, dword ptr [edi+28-14]      ; request->senders_ip
        mov     dword ptr [esi+38], eax

        pop     esi                             ; ESI -> reply ECB
        mov     edi, 1514                       ; Max Packet size
        xor     eax, eax                        ; Good packet
        mov     ecx, [esi].RPacketSize
        push    ebp
        call    EtherTSMFastProcessGetRCB
        pop     ebp
        jne     DriverSendReturnARP             ; Jump if no new ECB
        MSMReturnRCB                            ; Return newly allocated ECB

DriverSendReturnARP:
        pop     esi                             ; Restore send ECB
        inc     [ebp].MSMTxFreeCount            ; Add a send resource
        jmp     EtherTSMFastSendComplete        ; Otherwise service events.

DriverSend      endp

        extrn   DoEndOfInterrupt: near
        extrn   SetHardwareInterrupt: near
        extrn   ClearHardwareInterrupt: near
TestDriverISR   proc

        mov     ebp, OurAdapterDataSpace
        mov     ebx, [ebp].MSMDefaultVirtualBoard
        movzx   ecx, [ebx].MLIDInterrupt
        call    DoEndOfInterrupt

        inc     [ebp].GotInterrupt

        xor     eax, eax
        ret

TestDriverISR   endp
```

```
        subttl  -- DriverISR --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverISR, DPC/API/ISR )
;
; Name:         DriverISR
;
; Description:  This routine handles packet reception.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Destroyed
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Destroyed
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;               It is called at interrupt time.
;
; See Also:     MSM\MSMInterruptProcedure
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
        align   16
        public  DriverISR
DriverISR       proc
;
; /* Set the adapters ram ptr to the next rbd to receive from */
; outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
;
        DebugMessage    DEBUG_ISR_ALL, ISREnterMsg
        mov     edx, [ebp].IOMsgRamPtr                  ; MsgRamPtr I/O port
        mov     eax, [ebp].CurrentAdapterRBD            ; Next Adapter RBD
        shl     eax, 1                                  ;  * 2
        add     eax, RBD_BASE_ADDR                      ; add Base(0a000h)
        out     dx, ax                                  ; Set Adapter Ram Ptr
;
; /* Keep processing packets until no more are left */
; /*
;  * NOTE: We are assuming that anyone looping back to DriverISRLoop
;  * has set the MSR_RAM_PTR to the next RBD to examine.
; */
; while ((status = inport (bicd_base_addr + MSG_RAM)) & EMPTY)
;
DriverISRLoop:
        xor     eax, eax                                ; Clear upper status
```

```
        mov     edx, [ebp].IOMsgRam                     ; MsgRam I/O port
        in      ax, dx                                  ; Get status
        mov     [ebp].IntStatus, eax                    ; Save off RBD status

        test    eax, EMPTY                              ; This on ready?
        je      DriverISRExit                           ; Jump if its not ready

        inc     [ebp].BufferCount                       ; Used for watchdog
        DebugMessage1   DEBUG_ISR, DebugRBDReceived, [ebp].CurrentAdapterRBD
;
;   /* Jump if this is an error packet */
;   if (status & 0x8F)
;
        test    [ebp].IntStatus, STATUS_ERROR           ; Any error bits set?
        jne     DriverISRBadPacket                      ; Jump if not
;
;   /* Heres a good packet. See if we have a buffer for it. */
;   if (!global_pool[curr_rbd].buf_ptr)
;
        mov     esi, [ebp].CurrentECB                   ; Is this more of
        or      esi, esi                                ;  the last packet?
        jne     DriverISRAddToECB                       ; Jump if it is.
if TIMESTAMP
;       mov     al, 'r'
;       push    eax
;       call    DPCTimestamp
;       lea     esp, [esp + 4]
endif

        mov     esi,1514                                ; Max ECB size.
        call    MSMAllocateRCB                          ; Get an ECB
        or      eax, eax
        jne     DriverISRNoECB                          ; Jump if no ECB.

;
; !!!! Satelite header is 12 bytes, EII is 14 bytes.
; !!!! Add 2 to offset to prevent double copy of turbo internet packets.
;
        lea     edi, [esi+RPacketEnvelope+2]    ; EDI -> beginning
        mov     [esi].RPacketOffset, edi                ; Store into ECB
        mov     [esi].RPacketSize, 0                    ; Clear size
        mov     [ebp].CurrentECB, esi                   ; Store if split packet
        jmp     short DriverISRReadSize

DriverISRAddToECB:
        mov     edi, [esi].RPacketOffset
        add     edi, [esi].RPacketSize

DriverISRReadSize:
;
;   /* ESI(curr_rbd) will be used a lot. Let's try to keep it intact. */
;   /* Retrieve the length of the packet */
;   length = inport(bicd_base_addr + MSG_RAM);
;
        xor     eax, eax                                ; Clear upper word
        mov     edx, [ebp].IOMsgRam                     ; Msg Ram I/O port
        in      ax, dx                                  ; Get size of packet

        add     [esi].RPacketSize, eax                  ; Add to ECB size

        DebugMessage1   DEBUG_ISR, DebugRBDSize, eax

;
;   word_length = (length & 3) ? (length / 4) + 1 : (length/4);
;
```

```
        mov     ecx, eax                                ; ECX = packet length

        shr     ecx, 2                                  ; ECX = ECX / 4
        test    eax, 3                                  ; Any left over?
        jz      DriverISRCopyPacket                     ; Jump if not
        inc     ecx                                     ; read another dword

;;      shr     ecx, 1
;;      test    eax, 1
;;      jz      DriverISRCopyPacket
;;      inc     ecx
DriverISRCopyPacket:
;
;   outport(bicd_base_addr + AUTO_INC, curr_adap_rbd * stats.config.buf_size /
2);
;
        push    ecx
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = current adapter
RBD
;       mov     ecx, RBD_BUFFER_SIZE                    ; ECX = buffer size
        mov     ecx, RBD_BUFFER_SIZE / 2                ; ECX = buffer size
        mul     ecx                                     ; EAX = EAX * ECX
;       shr     eax, 1                                  ; EAX = EAX / 2
        mov     edx, [ebp].IOAutoInc                    ; Auto Inc I/O port
        out     dx, ax                                  ; Set adapter buffer ptr
        pop     ecx
;
;   Read from RxData port really fast(this was inline assembly)
;
        cld                                             ; Forward march!
        mov     edx, [ebp].IORxData                     ; Rx Data I/O port
        rep     insd                                    ; Repeat input to string
;;      rep     insw                                    ; Repeat input to string
;
;   /* Release an adapter buffer */
;   outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2 * curr_adap_rbd);
;   outport(bicd_base_addr + MSG_RAM, 0);
;   outport(bicd_base_addr + MSG_RAM, stats.config.buf_size);
;
        mov     edx, [ebp].IOMsgRamPtr                  ; Ram Ptr I/O port
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = adapter rbd
        shl     eax, 1                                  ; EAX = EAX * 2
        add     eax, RBD_BASE_ADDR                      ; EAX = EAX + base
        out     dx, ax                                  ; set ram ptr to adap rb
d

        mov     edx, [ebp].IOMsgRam                     ; Ram I/O port
        xor     eax, eax                                ; Clear status for reuse
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE                    ; buffer size
        out     dx, ax                                  ; set adapters buf size
;
;   /* Bump Current Adapter RBD value and make sure it wraps */
;   if (++curr_adap_rbd == stats.cnofig.adap_rbd_num)
;       curr_adap_rbd = 0;
;
        mov     eax, [ebp].CurrentAdapterRBD            ; EAX = current adap rbd
        inc     eax                                     ; Bump it
        cmp     eax, ADAP_RBD_NUM                       ; Did it go over?
        jb      DriverISRRBDWrap                        ; Jump if not
        xor     eax, eax                                ; Force it to wrap
DriverISRRBDWrap:
        mov     [ebp].CurrentAdapterRBD, eax            ; Save for later
        DebugMessage1   DEBUG_ISR_ALL, AdapterRBDMsg, eax
;
```

```
;   /* Now set ram ptr to next rbd in case we loop up to DriverISRLoop */
;   outport(bicd_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
;
        mov     edx, [ebp].IOMsgRamPtr
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
;
;   /* This was the first buffer. Save it off for unravel */
;       first_buffer_rbd = curr_rbd;
;       first_buffer = 0;
;       frame_buffers = 0;
;   }
;

;   if (!(status & EOF_BIT))
;   {

        test    [ebp].IntStatus, EOF_BIT                ; Last buffer?
        je      DriverISRLoop                           ; Jump if it wasn't
;
;   /* We have the last frame. Pass buffers to application. */
;   /*
;    * First lets see if we can match the packet address to a
;    * filter table address.
;    */
;   for (ii = 0; ii < BICDD_MAX_ADDR; ii++)
;    if (!fmemcmp( filter[ii].address, global_pool[first_buffer_rbd].buf_ptr, 6
)
;     && (filter[ii].channel != BICDD_NOT_USED)
;
        mov     [ebp].CurrentECB, 0                     ; Clear ECB flag
        lea     edi, [ebp].Filter                       ; EDI -> filter[0]
DriverISRFilterLoop:
        mov     eax, dword ptr [edi].FilterAddress+0    ; EAX = 1st filter addr
dword
        mov     edx, [esi].RPacketOffset                ; EDX -> buffer
        cmp     eax, [edx+0]                            ; 1st dword compare?
        jne     DriverISRFilterNext                     ; Jump if not
        mov     ax, word ptr [edi].FilterAddress+4      ; AX = last filter addr
word
        cmp     ax, [edx+4]                             ; last word compare?
        jne     DriverISRFilterNext                     ; Jump if not
;
; /* We have a winner! */
; /* EDI points to Filter entry */
;
        mov     eax, [edi].FilterChannel                ; EAX = channel
        cmp     eax, RBD_NOT_USED                       ; Valid channel?
        je      DriverISRFilterNext                     ; Jump if not

        DebugMessage6   DEBUG_ISR_ALL, FilterAddrMsg, [edx+0], [edx+1], [edx+2],
[edx+3], [edx+4], [edx+5]
;
; /* Lets set ESI to Rx Control entry */
;       cc = filter[ii].channel;
;
        lea     ebx, [ebp].RxControl[eax*8]             ; EBX -> Rx Control entr
y
;
;
;       /* Check whether length in the header matches the length rxed */
;       for (ll = 0, length = 0, mm = first_buffer-rbd; ll < frame_buffers; ll++
)
;       {
;
;
```

```
;       /* ECX = length = total length of all buffers */
;       p_length = (global_pool[first_buffer_rbd].buf_ptr) + 3;
;       if (((*p_length + 12) > length) || ((*p_length + 12) < (length - 16)))
;       {
;
        mov     ecx, [esi].RPacketSize                  ; ECX = hardware size
        mov     edx, [esi].RPacketOffset                ; EDX -> packet
        mov     edx, [edx+6]                            ; EDX = header length
        and     edx, 0ffffh                             ; we only need the word
        add     edx, 12                                 ; EDX = header length +
12
        cmp     edx, ecx                                ; *p_length+12 > length
        ja      DriverISRFilterSkipRBDsLen              ; Jump if it is
        sub     ecx, 16                                 ; ECX = length - 16
        cmp     edx, ecx                                ; *p_length+12 < length-
16
        jb      DriverISRFilterSkipRBDsLen              ; Jump if it is
;
;       filter[ii].total_count++;
;
        inc     [edi].FilterTotalCount                  ; Bump filter total coun
t
;
;       /* Check address seq numbers and update stats
;       p_seq_num = (global_pool[first_buffer_rbd].buf_ptr) + 2;
;
        mov     edx, [esi].RPacketOffset                ; EDX -> packet
        mov     eax, [edx+8]                            ; EAX = sequence number
;
;       if (*p_seq_num && filter[ii].seq_num && (*p_seq_num != filter[ii].seq_nu
m++)
;       {
;          filter[ii].seq_count++;
;          filter[ii].seq_num = *p_seq_num + 1;
;       }
;       else if (!*p_seq_num)
;          filter[ii].seq_num = 1;
;       else if (!filter[ii].seq_num)
;          filter[ii].seq = *p_seq_num + 1;
;
        or      eax, eax                                ; did header have a sequ
ence?
        je      DriverISRFilterNoPacketSeq              ; jump if not
        cmp     [edi].FilterSeqNum, 0                   ; does filter sequence e
xist?
        je      DriverISRFilterNoFilterSeq              ; jump if not
        cmp     eax, [edi].FilterSeqNum                 ; Do they match?
        jne     DriverISRFilterSeqNoMatch               ; Jump if not.
        inc     [edi].FilterSeqNum                      ; filter[ii].seq_num++
;
;       /* Discard frames if it's a "stats only" channel */
;       if (rx_cntl[cc].flags & BICDD_FLAGS_STATS_ONLY)
;       {
;
DriverISRFilterCallESR:
        mov     eax, [ebx].RxESR                        ; EAX -> channel ESR
        or      eax, eax
        je      DriverISRDidntWantECB                   ; Jump if no ESR
        cmp     eax, 0ffffffffh
        jne     DriverISRNotOurs

        public DriverISRInternet
DriverISRInternet:
;       INT_3
        lea     edi, [esi+RPacketEnvelope]              ; EDI -> EII header
        mov     ebx, [ebp].MSMDefaultVirtualBoard
```

```
        mov     ecx, [esi].RPacketOffset                        ; ECX -> Satelite header
        ;
        ; Make sure its not a data feed address. We don't know what to do
        ; with these yet.
        ;
        mov     al, [ecx+0]
        mov     ah, [ecx+2]
        and     al, 3
        cmp     al, 3
        je      MightBeDataStreamPacket
        and     al, 1
        cmp     al, 1
        jne     NotDataStreamPacket
MightBeDataStreamPacket:
        cmp     ah, 0ffh
        jne     NotDataStreamPacket
        ;
        ; Data stream, go into debugger.
        ;
        INT_3                                                   ; Break into debugger
        MSMReturnRCB                                            ; Return it
        jmp     DriverISRLoop                                   ; Get next packet


NotDataStreamPacket:
        ;
        ; Fill in EII Destination with our node address
        ;
        mov     eax, dword ptr [ebx].MLIDNodeAddress
        mov     [edi+0], eax
        mov     ax, word ptr [ebx].MLIDNodeAddress+4
        mov     [edi+4], ax
        ;
        ; Fill in EII Source Address with (0x0a 0x0a 0x0a 0x0a 0x0a 0x0a)
        ;
;       mov     word ptr [edi+6], 0a0ah
;       mov     dword ptr [edi+8], 0a0a0a0ah
        mov     word ptr [edi+6], 0606h
        mov     dword ptr [edi+8], 06060606h
        ;
        ; Fill in EII type field with IP type (0x08 0x00)
        ;
        mov     word ptr [edi+12], 0008         ; Force IP PID(800h hi/lo) since
 DPC is unaware of it

        lea     ecx, [esi + RFragmentCount]
        push    ecx
        call    [DPCRxFrame]
        add     esp, 4

        movzx   ecx, word ptr [esi + RPacketEnvelope + 14 + 2]
        xchg    ch, cl
        add     ecx, 14
        cmp     ecx, 3ch
        jae     RxPacketIsPadded
        mov     ecx, 3ch
RxPacketIsPadded:

;
; vvv DEBUG
        mov     eax, ecx
        cmp     eax, 400
        jb      DebugRxExit

        cmp     eax, [ebp].LargestRx
        jbe     DebugRxAve
        mov     [ebp].LargestRx, eax

DebugRxAve:
        inc     [ebp].NumberLargeRx
        add     eax, [ebp].TotalLargeRx
        mov     edi, [ebp].NumberLargeRx
        mov     [ebp].TotalLargeRx, eax
        xor     edx, edx
        div     edi
        mov     [ebp].AveLargeRx, eax

DebugRxExit:

; ^^^ DEBUG
;
        mov     edi, 1514                       ; Max Packet size
if TIMESTAMP
;       push    ecx
;       mov     al, 'R'
;       push    eax
;       call    DPCTimestamp
;       lea     esp, [esp + 4]
;       pop     ecx
endif
        xor     eax, eax                        ; Good packet

        push    ebp
        call    EtherTSMFastProcessGetRCB
        pop     ebp
        jne     DriverISRLoop                                   ; Jump i
f no ecb returned
        jmp     DriverISRDidntWantECB           ; Give it back. We don't store t
hese

DriverISRNotOurs:
        push    esi                             ; Parm0 = ECB
        call    eax                             ; call ESR
        pop     esi                             ; clean up stack
        or      eax, eax                        ; did they keep it?
        je      DriverISRLoop                   ; jump if they did.

DriverISRDidntWantECB:
        MSMReturnRCB                            ; Return it
        jmp     DriverISRLoop                   ; Get next packet


DriverISRFilterNext:
        add     edi, size FilterStruct
        lea     eax, [ebp].Filter[MAX_ADDR * size FilterStruct]
        cmp     edi, eax
        jbe     DriverISRFilterLoop
;
; Couldn't find filter address. Clean up.
;

        MSMReturnRCB                            ; Give ECB back.
        DebugMessage6   DEBUG_ISR_ALL, FilterNone, [edx+0], [edx+1], [edx+2], [e
dx+3], [edx+4], [edx+5]
        jmp     DriverISRLoop


DriverISRFilterSeqNoMatch:
        inc     eax
        inc     [edi].FilterSeqCount
        mov     [edi].FilterSeqNum, eax
        jmp     DriverISRFilterCallESR
```

```
DriverISRFilterNoFilterSeq:
        inc     eax
        mov     [edi].FilterSeqNum, eax
        jmp     DriverISRFilterCallESR

DriverISRFilterNoPacketSeq:
        mov     [edi].FilterSeqNum, 1
        jmp     DriverISRFilterCallESR

DriverISRFilterSkipRBDsLen:
        MSMReturnRCB
        DebugMessage2   DEBUG_ISR_ALL, FilterRBDLen, ecx, edx
        jmp     DriverISRLoop

DriverISRNoECB:
        DebugMessage    DEBUG_ISR, NoECBMsg
        jmp     DriverISRBadNextRBD

DriverISRBadPacket:
        mov     esi, [ebp].IntStatus
        test    esi, FRAMING_ERR
        je      DriverISRCheckAbort

        DebugMessage    DEBUG_ISR, FramingErrMsg

DriverISRCheckAbort:
        test    esi, ABORT
        je      DriverISRCheckAlign

        DebugMessage    DEBUG_ISR, AbortMsg

DriverISRCheckAlign:
        test    esi, ALIGN_ERR
        je      DriverISRCheckOverrun

        DebugMessage    DEBUG_ISR, AlignErrMsg

DriverISRCheckOverrun:
        test    esi, OVERRUN_ERR
        je      DriverISRCheckDES

        DebugMessage    DEBUG_ISR, OverrunErrMsg

DriverISRCheckDES:
        test    esi, DES_ERR
        je      DriverISRCheckCRC

        DebugMessage    DEBUG_ISR, DESErrMsg

DriverISRCheckCRC:
        test    esi, CRC_ERR
        je      DriverISRErrorStats

        DebugMessage    DEBUG_ISR, CRCErrMsg

DriverISRErrorStats:

        DebugMessage    DEBUG_ISR, ReturnMsg

DriverISRBadNextRBD:
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, [ebp].CurrentAdapterRBD
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
        mov     edx, [ebp].IOMsgRam
        xor     eax, eax
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax

        mov     eax, [ebp].CurrentAdapterRBD
        inc     eax
        cmp     eax, ADAP_RBD_NUM
        jb      DriverISRBadRBDWrap
        xor     eax, eax
DriverISRBadRBDWrap:
        mov     [ebp].CurrentAdapterRBD, eax

        DebugMessage1   DEBUG_ISR_ALL, AdapterRBDMsg, eax
        mov     edx, [ebp].IOMsgRamPtr
        shl     eax, 1
        add     eax, RBD_BASE_ADDR
        out     dx, ax
        jmp     DriverISRLoop

DriverISRExit:
        mov     edx, [ebp].IOMsgRamPtr
        mov     eax, 0c3a0h
        out     dx, ax

        mov     eax, [ebp].CurrentAdapterRBD
        mov     edx, [ebp].IOMsgRam
        out     dx, ax

        DebugMessage1   DEBUG_ISR_ALL, ISRExitMsg, eax

        mov     edx, [ebp].IOStatus
        in      ax, dx

        ret

DriverISR       endp
        subttl  -- DriverDisableInterrupt --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverDisableInterrupt, DPC/API/DISINT )
;
; Name:         DriverDisableInterrupt
;
; Description:  This routine will disable the adapters ability to
;               interrupt the host.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
```

```
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;
; See Also:     DriverEnableInterrupt
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
        align   16
DriverDisableInterrupt  proc

        xor     eax, eax
        ret

DriverDisableInterrupt  endp
        subttl  -- DriverEnableInterrupt --
        page
;
;*****************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverEnableInterrupt, DPC/API/ENINT )
;
; Name:         DriverEnableInterrupt
;
; Description:  This routine will enable the adapters ability to
;               interrupt the host.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM.
;
; See Also:     DriverDisableInterrupt
;
; END_MANUAL_ENTRY
;
;*****************************************************************************/
;
        align   16
```

```
DriverEnableInterrupt   proc

        ret

DriverEnableInterrupt   endp
        public  DriverReset
        subttl  -- DriverReset --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverReset, DPC/API/RESET )
;
; Name:         DriverReset
;
; Description:  This routine will reset and initialize the NIC.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     N/A
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     0 if successful(otherwise points to error message)
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Destroyed
;               EDI     Destroyed
;
;               Flags:
;
;               Note:   Interrupts disabled.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process time.
;
; See Also:     ETHERTSM\EtherTSMReset
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
DriverReset     proc    near

        inc     [ebp].AdapterResetCount         ; Increment stat counter.

        xor     eax, eax
        ret

DriverReset     endp

DefaultRxFrame  proc

        ret

DefaultRxFrame  endp

        extrn   LSLGetStackIDFromName: near
ProtocolBindEvent       proc
```

```
        CPush

        lea     edx, IPName
        call    LSLGetStackIDFromName   ; Return Stack ID in EBX
        or      eax, eax
        jne     short ProtocolBindExit

        mov     esi, [esp + Parm0]
        cmp     [esi+4], ebx                    ; IP Stack?
        jne     short ProtocolBindExit  ; Nope
        mov     edx, [esi]                      ; EDX = Bound board number
        mov     ebp, OurAdapterDataSpace

        xor     ecx, ecx
ProtocolBindLoop:
        mov     ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or      ebx, ebx
        jz      ProtocolBindNext

        cmp     [ebx].MLIDBoardNumber, dx
        jne     ProtocolBindNext

        mov     eax, 1514
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

ProtocolBindNext:
        inc     ecx
        cmp     ecx, 4
        jb      ProtocolBindLoop
ProtocolBindExit:
        CPop
        ret

ProtocolBindEvent       endp

ProtocolUnbindEvent     proc

        CPush

        lea     edx, IPName
        call    LSLGetStackIDFromName   ; Return Stack ID in EBX
        or      eax, eax
        jne     short ProtocolUnbindExit

        mov     esi, [esp + Parm0]
        cmp     [esi+4], ebx                    ; IP Stack?
        jne     short ProtocolBindExit  ; Nope
        mov     edx, [esi]                      ; EDX = Bound Board Number
        mov     ebp, OurAdapterDataSpace

        xor     ecx, ecx
ProtocolUnbindLoop:
        mov     ebx, [ebp+MSMVirtualBoardLink][ecx*4]
        or      ebx, ebx
        jz      ProtocolUnbindNext

        cmp     [ebx].MLIDBoardNumber, dx
        jne     ProtocolUnbindNext

        mov     eax, 1494
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

ProtocolUnbindNext:
        inc     ecx
        cmp     ecx, 4
        jb      ProtocolUnbindLoop
ProtocolUnbindExit:
        CPop
        ret

ProtocolUnbindEvent     endp


        subttl  -- DriverInit --
        page
;
;**************************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverInit, DPC/API/INIT )
;
; Name:         DriverInit
;
; Description:  This routine will call EtherTSMRegisterHSM,
;               MSMParseDriverParameters, MSMRegisterHardwareOptions,
;               MSMSetHardwareInterrupt, MSMRegisterMLID, initialize
;               variables in the Adapter Data Space and reset/initialize
;               the card.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     N/A
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are enabled.
;
; On Return:    EAX     0 if successful(otherwise it points to error message)
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the OS at load time.
;               It is called at process time.
;
; See Also:     MSM\MSMParseDriverParameters
;               MSM\MSMRegisterHardwareOptions
;               MSM\MSMSetHardwareInterrupts
;               MSM\MSMRegisterMLID
;               MSM\MSMScheduleIntTimeCallBack
;               MSM\MSMScheduleAESCallBack
;               MSM\MSMEnablePolling
;               DriverReset
;
; END_MANUAL_ENTRY
;
;**************************************************************************/
;
```

```
        extrn   RegisterForEventNotification: near
        extrn   UnRegisterEventNotification: near

DriverInit      proc

        CPush
if TIMESTAMP
        lea     eax, DPCTB
        mov     timestamp_begin, eax
        mov     timestamp_index, eax
        add     eax, TIMESTAMP_BUFFER_SIZE
        mov     timestamp_end, eax
endif
;
;*******************************************************************\
;                                                                   *
; Fill in Driver Parameter Block fields.                            *
;                                                                   *
;*******************************************************************/
;
        mov     DriverStackPointer, esp                 ; Fill in stack ->.

        lea     esi, DriverParameterBlock               ; ESI -> Parm block.
        call    EtherTSMRegisterHSM                     ; Get EBX.
        jnz     DriverInitError                         ; Jump if error.
;
; Yuck' We'll have to adjust the receive size down, since
; Hughes can't handle full 1500 byte packets with tunneling.
;
        mov     [ebx].MLIDMaximumSize, 1494
;
;*******************************************************************\
;                                                                   *
; EBX -> Frame Data Space(Config Table).                            *
; Let MSM Parse the command line.                                   *
;                                                                   *
;*******************************************************************/
;
        mov     GlobalRxFreq, DEFAULT_RX_FREQ

        mov     eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRE
SS
        lea     ecx, AdapterOptions
        call    MSMParseDriverParameters
        jnz     DriverInitError                         ; Jump if error.
;
;*******************************************************************\
;                                                                   *
; Let MSM Register the hardware options.                            *
;                                                                   *
;*******************************************************************/
;
        call    MSMRegisterHardwareOptions
        cmp     eax, 1                                  ; Error Registering?
        ja      DriverInitError                         ; Jump if so.
        je      DriverInitExit                          ; Skip if new frame.

        mov     OurAdapterDataSpace, ebp                        ; Save for later
        mov     DPCRxFrame, offset DefaultRxFrame
;
; Get a timer resource tag so that we can delay ourselves.
;
        push    TimerSignature
        push    offset TimerDesc
      \ push    DriverModuleHandle
        call    AllocateResourceTag
        lea     esp, [esp + (3 * 4)]
        mov     [ebp].TimerTag, eax
        or      eax, eax
        lea     eax, ErrorAllocatingRTagMessage
        je      DriverInitErrorReturn
;
; Get a resource tag for interrupts.
;
        push    InterruptSignature              ; Pass signature.
        push    offset InterruptRTagMessage     ; Pass Message.
        push    DriverModuleHandle              ; Pass Module handle.
        call    AllocateResourceTag             ; Allocate RTag.
        add     esp, (3 * 4)                    ; Clean up stack.
        mov     [ebp].ISRTag, eax               ; Save RTag.
        or      eax, eax                        ; RTag returned?
        lea     eax, ErrorAllocatingRTagMessage ; EAX -> Error mess.
        je      DriverInitErrorReturn       ; Jump if not.
;
; Get a resource tag for event notification.
;
        push    EventSignature              ; Pass signature.
        push    offset EventRTagMessage     ; Pass Message.
        push    DriverModuleHandle              ; Pass Module handle.
        call    AllocateResourceTag             ; Allocate RTag.
        add     esp, (3 * 4)                    ; Clean up stack.
        mov     [ebp].EventTag, eax             ; Save RTag.
        or      eax, eax                        ; RTag returned?
        lea     eax, ErrorAllocatingRTagMessage ; EAX -> Error mess.
        je      DriverInitErrorReturn       ; Jump if not.
;
; Register for protocol bind event.
;
        mov     eax, [ebp].EventTag
        push    offset ProtocolBindEvent            ; Pass procedure.
        push    0                                   ; No Warning proc.
        push    EVENT_PRIORITY_OS                   ; Priority level.
        push    EVENT_PROTOCOL_BIND                 ; Event type.
        push    eax                                 ; Pass resource tag.
        call    RegisterForEventNotification    ; Register Event.
        add     esp, (4 * 5)
        mov     [ebp].ProtocolBindID, eax                   ; Save Event ID.
        or      eax, eax                            ; Registered?
        je      DriverInitNoBindEvent       ; Jump if not.

        mov     eax, [ebp].EventTag
        push    offset ProtocolUnbindEvent          ; Pass procedure.
        push    0                                   ; No Warning proc.
        push    EVENT_PRIORITY_OS                   ; Priority level.
        push    EVENT_PROTOCOL_UNBIND       ; Event type.
        push    eax                                 ; Pass resource tag.
        call    RegisterForEventNotification    ; Register Event.
        add     esp, (4 * 5)
        mov     [ebp].ProtocolUnbindID, eax         ; Save Event ID.
        or      eax, eax                            ; Registered?
        jne     DriverInitSetPorts                  ; Jump if not.

DriverInitNoBindEvent:
        mov     eax, 1494                           ; Just take the max down
        mov     [ebx].MLIDMaximumSize, eax
        sub     eax, 14
        mov     [ebx].MLIDMaxRecvSize, eax
        mov     [ebx].MLIDRecvSize, eax

DriverInitSetPorts:
;
;*************************************************************\
```

```
;                                                                *
; Set and check the adapters base I/O.                           *
;                                                                *
;***************************************************************/
;
        movzx   ecx, [ebx].MLIDIOPortsAndLengths
        mov     [ebp].IORxData, ecx                     ; Rx Data port = base +
0
        add     ecx, 2
        mov     [ebp].IOAutoInc, ecx                    ; Auto Inc port = base +
 2
        add     ecx, 2
        mov     [ebp].IOStatus, ecx                     ; Status port = base + 4
        add     ecx, 2
        mov     [ebp].IOControl, ecx                    ; Control port = base +
6
        add     ecx, 2
        mov     [ebp].IOMsgRamPtr, ecx                  ; Msg Ram Ptr = base + 8
        add     ecx, 2
        mov     [ebp].IOMsgRam, ecx                     ; Msg Ram = base + 10
        add     ecx, 2
        mov     [ebp].IORbdBufLen, ecx                  ; RDB buf Len = base + 1
2
        add     ecx, 2
        mov     [ebp].IORbdNum, ecx                     ; RDB Num = base + 14
        add     ecx, 2
        mov     [ebp].IOBtrControlAddr, ecx             ; BTR Control Addr = bas
e + 10h + 0h
        add     ecx, 2
        mov     [ebp].IOAfcControlAddr, ecx             ; AFC Control Addr = bas
e + 10h + 2h
        add     ecx, 2
        mov     [ebp].IOBitDetControlAddr, ecx          ; Bit Det Control Addr =
 base + 10h + 4h
        add     ecx, 2
        mov     [ebp].IOAgcFirControlAddr, ecx          ; Agc Fir Control Addr =
 base + 10h + 6h
        add     ecx, 2
        mov     [ebp].IOCrlkThrLowAddr, ecx             ; Crlk Thr Low Addr = ba
se + 10h + 8h
        add     ecx, 2
        mov     [ebp].IOCthAddr, ecx                    ; Cth Addr = base + 10h
+ 0ah
        add     ecx, 2
        mov     [ebp].IOGateCountHighAddr, ecx          ; Gate Count High Addr =
 base + 10h + 0ch
        add     ecx, 2
        mov     [ebp].IOCountNomLowAddr, ecx            ; Count Nom Low Addr = b
ase + 10h + 0eh
        add     ecx, 2
        mov     [ebp].IOCountNomHighAddr, ecx           ; Count Nom High Addr =
base + 10h + 10h
        add     ecx, 2
        mov     [ebp].IOCountDeltaAddr, ecx             ; Count Delta Addr = bas
e + 10h + 12h
        add     ecx, 2
        mov     [ebp].IOSweepRateAddr, ecx              ; Sweep Rate Addr = base
 + 10h + 14h
        add     ecx, 2
        mov     [ebp].IOCrlkControlAddr, ecx            ; Crlk Control Addr = ba
se + 10h + 16h
        add     ecx, 2
        mov     [ebp].IOSynthSerControlAddr, ecx        ; Synth Ser Control Addr
 = base + 10h + 18h
        add     ecx, 2
        mov     [ebp].IOSpareIOControlAddr, ecx         ; Spare IO Control Addr
```

```
= base + 10h + 1ah
        add     ecx, 2
        mov     [ebp].IODaAdOffsetControlAddr, ecx      ; IO DA Offset Control A
ddr = base + 10h + 1ch
        add     ecx, 2
        mov     [ebp].IOUnitIDAddr, ecx                 ; Unit ID Addr = base +
10h + 1eh
        add     ecx, 2

        movzx   ecx, [ebx].MLIDIOPortsAndLengths
        mov     al, 0bh                                 ; AL = 0bh
        cmp     ecx, 100h                               ; I/O port 100h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 0ah
        cmp     ecx, 140h                               ; I/O port 140h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 9
        cmp     ecx, 180h                               ; I/O port 180h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 8
        cmp     ecx, 1c0h                               ; I/O port 1c0h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 7
        cmp     ecx, 200h                               ; I/O port 200h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 6
        cmp     ecx, 240h                               ; I/O port 240h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 5
        cmp     ecx, 280h                               ; I/O port 280h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 4
        cmp     ecx, 2c0h                               ; I/O port 2c0h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 3
        cmp     ecx, 300h                               ; I/O port 300h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 2
        cmp     ecx, 340h                               ; I/O port 340h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 1
        cmp     ecx, 380h                               ; I/O port 380h?
        je      SetPort                                 ; yes
        dec     al                                      ; AL = 0
SetPort:
        mov     dx, 279h
        out     dx, al                                  ; Set Base I/O port
;
; Lets Reset the adapter.
;
        push    eax

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_MRESET
        out     dx, ax

        mov     eax, [ebp].TimerTag
        push    eax
        push    2
        call    DelayMyself
        add     esp, (2 * 4)

        mov     edx, [ebp].IOControl
        mov     eax, 0
        out     dx, ax
```

```
        pop     eax
        mov     dx, 279h
        out     dx, al
;
; Make sure that we can set spare output
;
        mov     edx, [ebp].IOControl                    ; Set Spare Output
        mov     eax, CNTL_SOUTPUT
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx
        test    eax, STAT_SOUTPUT
        mov     eax, offset MsgIOSetFailed
        je      DriverInitErrorReturn
;
; Make sure that we can clear spare output
;
        mov     edx, [ebp].IOControl                    ; Clear Spare Output
        mov     eax, 0
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx
        test    eax, STAT_SOUTPUT
        mov     eax, offset MsgIOClearFailed
        jne     DriverInitErrorReturn
;
; If no node override, default it.
;
        cmp     dword ptr [ebx].MLIDNodeAddress, -1
        jnz     short NodeIsSet
        mov     [ebx].MLIDNodeAddress+0, 00h
        mov     [ebx].MLIDNodeAddress+1, 80h
        mov     [ebx].MLIDNodeAddress+2, 0aeh
        mov     [ebx].MLIDNodeAddress+3, 00h
        mov     [ebx].MLIDNodeAddress+4, 00h
        mov     [ebx].MLIDNodeAddress+5, 01h

NodeIsSet:
;
;***************************************************************\
;                                                               *
; Download the MIPS code to the adatper.                        *
;                                                               *
;***************************************************************/
;
        mov     edx, [ebp].IOControl            ; Set to auto-increment mode
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     ecx, MipsCodeSize               ; Number of words to copy
        mov     edx, [ebp].IOMsgRam             ; adapter ram port
        lea     esi, MipsCode                   ; ESI -> mips code
        cld
CopyToAdapterLoop:
        lodsw                                   ; Get Mips Word
        out     dx, ax                          ; Send it to adapter
        dec     ecx
        jnz     CopyToAdapterLoop
;
; Verify the download by reading the data back
;
        mov     edx, [ebp].IOControl            ; Set to auto-increment mode
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRamPtr          ; Set Adapter Ram Address
        xor     eax, eax                        ;  to zero
        out     dx, ax

        mov     ecx, MipsCodeSize               ; Number of words to copy
        mov     edx, [ebp].IOMsgRam             ; adapter ram port
        lea     esi, MipsCode                   ; ESI -> mips code
        cld
VerifyAdapterLoop:
        xor     eax, eax
        lodsw                                   ; Get Mips Word
        mov     edi, eax
        in      ax, dx                          ; Send it to adapter
        cmp     edi, eax
        lea     eax, MsgBadRAM
        jne     DriverInitErrorReturn
        dec     ecx
        jnz     VerifyAdapterLoop
;
;*************************************************************\
;                                                             *
; Register our interrupt handler with the OS.                 *
;                                                             *
;*************************************************************/
;
        mov     edx, [ebp].IOStatus             ; Clear old errors
        in      ax, dx
;
; Set RBD base address
;
        mov     edx, [ebp].IORbdBase
        mov     eax, RBD_BASE_ADDR
        out     dx, ax
        mov     edx, [ebp].IOMsgRamPtr
        out     dx, ax

        mov     edx, [ebp].IORbdNum
        mov     eax, ADAP_RBD_NUM
        out     dx, ax
        mov     ecx, eax

        mov     edx, [ebp].IORbdBufLen
        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_AUTO_INC
        out     dx, ax

        mov     edx, [ebp].IOMsgRam
SetupBuffersLoop:
        xor     eax, eax
        out     dx, ax

        mov     eax, RBD_BUFFER_SIZE
        out     dx, ax
```

```
        dec     ecx
        jnz     SetupBuffersLoop
;
; Enable the adapter.
;
        movzx   ecx, [ebx].MLIDInterrupt
        mov     esi, CNTL_IRQ3
        mov     edx, 8
        cmp     ecx, 8h
        je      EnableDPC
        mov     esi, CNTL_IRQ4
        mov     edx, 10h
        cmp     ecx, 4
        je      EnableDPC
        mov     esi, CNTL_IRQ5
        mov     edx, 20h
        cmp     ecx, 5
        je      EnableDPC
        mov     esi, CNTL_IRQ9
        mov     edx, 2h
        cmp     ecx, 8
        je      EnableDPC
        mov     esi, CNTL_IRQ10
        mov     edx, 4h
        cmp     ecx, 10
        je      EnableDPC
        mov     esi, CNTL_IRQ11
        mov     edx, 8h
        cmp     ecx, 11
        je      EnableDPC
        mov     esi, CNTL_IRQ12
        mov     edx, 10h
        cmp     ecx, 12
        je      EnableDPC
        mov     esi, CNTL_IRQ15
        mov     edx, 80h
EnableDPC:
        mov     [ebp].PicMask, edx
        not     edx
        mov     [ebp].PicUnMask, edx
        mov     [ebp].PicAddress, 21h
        cmp     [ebx].MLIDInterrupt, 8
        jb      ClearOurInterrupt
        mov     [ebp].PicAddress, 0a1h
ClearOurInterrupt:
        mov     edx, [ebp].PicAddress
        in      al, dx
        and     eax, [ebp].PicUnMask
        out     dx, al

        mov     eax, esi
        mov     edx, [ebp].IOControl
        or      eax, CNTL_RX_EN OR CNTL_CPU_EN OR CNTL_INT_EN OR CNTL_SNGL_INT O
R CNTL_AUTO_INC OR CNTL_SOUTPUT
        out     dx, ax
        mov     [ebp].IOEnableValue, eax

        cmp     DebugMask, 0
        je      OpenScreenExit
        push    4e524353h                       ; 'NRCS'
        lea     eax, ScreenResourceName
        push    eax
        push    DriverModuleHandle
        call    AllocateResourceTag
        lea     esp, [esp + (3 * 4)]
        or      eax, eax
        je      OpenScreenExit

        mov     ScreenRTag, eax

        push    offset DPCScreen                ; Location of Handle storage
        push    eax                             ; Screen Resource Tag
        push    offset ScreenName               ; Name of screen object
        call    OpenScreen
        lea     esp, [esp + (3 * 4)]
        or      eax, eax
        jne     OpenScreenExit

        push    offset NLMName
        push    0
        push    DriverModuleHandle
        call    GetNLMNames
        lea     esp, [esp + (3 * 4)]

        push    0
        push    offset Day
        push    offset Month
        push    offset Year
        push    0
        push    offset MinorVer
        push    offset MajorVer
        push    DriverModuleHandle
        call    GetNLMVersionInfo
        lea     esp, [esp + (8 * 4)]

        push    Year
        push    Day
        push    Month
        push    MinorVer
        push    MajorVer
        push    offset NLMName
        push    offset DebugScreenHeader
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (8 * 4)]

OpenScreenExit:
;
;*************************************************************\
;                                                             *
; Initialize RxControl and Filter entries.                    *
;                                                             *
;*************************************************************/
;
        lea     edi, [ebp].RxControl
        mov     ecx, MAX_CHAN
        xor     eax, eax
        mov     edx, RBD_NOT_USED
InitRxControlLoop:
        mov     [edi].RxChannel, edx
        mov     [edi].RxESR, eax
        add     edi, size RX_CNTL
        dec     ecx
        jne     InitRxControlLoop

        lea     edi, [ebp].Filter
        mov     ecx, MAX_ADDR
InitFilterLoop:
        mov     [edi].FilterChannel, edx
        mov     [edi].FilterTotalCount, eax
        mov     [edi].FilterSeqCount, eax
        mov     [edi].FilterSeqNum, eax
```

```
        add     edi, size FilterStruct
        dec     ecx
        jne     InitFilterLoop
;
;*************************************************************\
;
; Test Interrupts.
;
;*************************************************************/
;
        ;
        ; Set ISR to test routine.
        ;
        mov     [ebp].GotInterrupt, 0   ; Clear test flag.
        mov     ecx, [ebp].ISRTag               ; ECX = ISR resource tag.
        push    0                               ; No ExtraEOIFlag offset.
        push    0                               ; ShareFlag.
        push    0                               ; End of chain flag.
        push    ecx                             ; ISR Resource tag.
        lea     eax, TestDriverISR
        push    eax                             ; ISR Entry point.
        movzx   eax, [ebx].MLIDInterrupt        ; EAX = Interrupt number.
        push    eax                             ; Interrupt Number.
        call    SetHardwareInterrupt            ; Set interrupt.
        lea     esp, [esp + (6 * 4)]            ; Restore stack.
        or      eax, eax                        ; Error setting interrupt?
        lea     eax, BadISRMsg          ; EAX -> Error Message.
        jnz     DriverInitErrorReturn   ; Exit if so.

        mov     edx, [ebp].IOControl
        mov     eax, [ebp].IOEnableValue
        or      eax, CNTL_FORCEINT
        out     dx, ax

        sti
        mov     eax, [ebp].TimerTag
        push    eax
        push    18
        call    DelayMyself
        add     esp, (2 * 4)
        cli

        movzx   eax, [ebx].MLIDInterrupt
        lea     edx, TestDriverISR
        push    edx                                     ; Pass ISR Entry point.
        push    eax                                     ; Pass Interrupt #.
        call    ClearHardwareInterrupt          ; Give interrupt back.
        lea     esp, [esp + (2 * 4)]                    ; Clean up stack.

        mov     eax, [ebp].IOEnableValue
        out     dx, ax

        lea     eax, NoInterruptMsg             ; Error message.
        cmp     [ebp].GotInterrupt, 0   ; Did we get it?
        je      DriverInitErrorReturn   ; Jump if not.
;
;*************************************************************\
;                                                             *
; EBX -> Frame Data Space(Config Table).                      *
; EBP -> Adapter Data Space.                                  *
;                                                             *
; Let MSM Set Hardware Interrupt.                             *
;                                                             *
;*************************************************************/
;
        call    MSMSetHardwareInterrupt
        jnz     DriverInitError                         ; Jump if error.
;
;*****************************************************************\
;                                                                 *
; Set TxFreeCount to make TSM happy.                              *
;                                                                 *
;*****************************************************************/
;
        mov     [ebp].MSMTxFreeCount, 32                ; Allow 32 transmits sim
ultaneously.

        mov     eax, 1                          ; Schedule call back in 18 ticks
.
        call    MSMScheduleIntTimeCallBack
        jnz     DriverInitError         ; Jump if error.

        call    DriverReset                     ; Initialize NIC.
        jnz     DriverInitErrorReturn   ; Exit if error reseting.

        mov     [ebp].FirstTimeInit, 0          ; Disable DriverReset from
                                                ; testing the hardware again.

        dec     [ebp].AdapterResetCount         ; Adjust reset count.

        call    MSMRegisterMLID         ; Register MLID.
        jnz     DriverInitError         ; Jump if error.

        ; Lets see if the adapter is locked up.

        mov     eax, [ebp].TimerTag
        push    eax
        push    18
        call    DelayMyself
        add     esp, (2 * 4)

        call    RefreshMipsStats
        test    [ebp].MipsRxEnables, 8000000h   ; This shouldn't be big
        lea     eax, LockedAdapterMsg
        jne     DriverInitErrorReturn

        cmp     DebugMask, 0
        je      DriverInitExit
        movzx   eax, [ebx].MLIDInterrupt
        push    eax
        movzx   eax, [ebx].MLIDIOPortsAndLengths
        push    eax
        push    offset DebugInitOK
        push    DPCScreen
        call    OutputToScreen
        lea     esp, [esp + (4 * 4)]

DriverInitExit:
        mov     [ebx].MLIDMaxRecvSize, 1400
        xor     eax, eax
        CPop
        ret

DriverInitErrorReturn:
        push    eax                             ; Save error message.
        call    MSMReturnDriverResources        ; Return resources.
        mov     eax, DPCScreen
        or      eax, eax
        je      DriverInitErrorScreenClosed
        push    eax
        call    CloseScreen
        lea     esp, [esp + (1 * 4)]
```

```
        mov     DPCScreen, 0
DriverInitErrorScreenClosed:
        pop     eax                                     ; EAX -> Error message.

DriverInitError:
        mov     esi, eax                                ; ESI -> Error message.
        call    MSMPrintString                          ; Display message

        or      eax, 1                                  ; Do not load return code.
        CPop

        ret

DriverInit      endp
        subttl  -- DriverShutdown --
        page
;
;*******************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverShutdown, DPC/API/SHUTDOWN )
;
; Name:         DriverShutdown
;
; Description:  This routine will turn off the NIC.
;
; On Entry:     EAX     N/A
;               EBX     @ Frame Data Space
;               ECX     0 if Permanent Shutdown
;               EDX     N/A
;               EBP     @ Adapter Data Space
;               ESI     N/A
;               EDI     N/A
;
;               Note:   Interrupts are disabled.
;
; On Return:    EAX     0 if successful
;               EBX     Preserved
;               ECX     Preserved
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note:   Interrupts preserved.
;
; Remarks:      This routine is called by the MSM media module.
;               It is called at process time.
;
; See Also:     ETHERTSM\EtherTSMShutdown
;
; END_MANUAL_ENTRY
;
;*******************************************************************/
;
DriverShutdown  proc

        or      ecx, ecx
        jne     DriverShutdownAdapter
        mov     eax, [ebp].AgentRemoveRoutine
        or      eax, eax
        je      DriverShutdownAdapter

        call    eax
        mov     [ebp].AgentRemoveRoutine, 0

DriverShutdownAdapter:

        pushfd
        cli
        mov     edx, [ebp].IOControl
        xor     eax, eax
        out     dx, ax

        mov     edx, [ebp].IOStatus
        in      ax, dx

        or      ecx, ecx
        jne     DriverShutdownExit

        mov     edx, [ebp].IOControl
        mov     eax, CNTL_MRESET
        out     dx, ax

        mov     eax, DPCScreen
        or      eax, eax
        je      DriverShutdownScreenClosed
        push    eax
        call    CloseScreen
        lea     esp, [esp + (1 * 4)]
        mov     DPCScreen, 0
DriverShutdownScreenClosed:

        mov     eax, [ebp].ProtocolBindID
        or      eax, eax
        je      DriverShutdownExit
        push    eax                             ; Pass Event ID.
        call    UnRegisterEventNotification     ; Unregister event.
        add     esp, (1 * 4)                    ; Clean up stack.

        mov     eax, [ebp].ProtocolUnbindID
        or      eax, eax
        je      DriverShutdownExit
        push    eax                             ; Pass Event ID.
        call    UnRegisterEventNotification     ; Unregister event.
        add     esp, (1 * 4)                    ; Clean up stack.

DriverShutdownExit:
        popfd
        xor     eax, eax                        ; Good Return code.
        ret

DriverShutdown  endp
        subttl  -- DriverRemove --
        page
;
;*********************************************************************\
;
; BEGIN_MANUAL_ENTRY( DriverRemove, DPC/API/REMOVE )
;
; Name:         DriverRemove
;
; Description:  This routine call the MSM to return our resources.
;
; On Entry:     EAX     N/A
;               EBX     N/A
;               ECX     N/A
;               EDX     N/A
;               EBP     N/A
;               ESI     N/A
```

```
;               EDI     N/A
;
;               Note:   Interrupts are in any state.
;
; On Return:    EAX     Destroyed
;               EBX     Preserved
;               ECX     Destroyed
;               EDX     Destroyed
;               EBP     Preserved
;               ESI     Preserved
;               EDI     Preserved
;
;               Flags:
;
;               Note·   Interrupts preserved.
;
; Remarks:      This routine is called by the OS at unload.
;               It is called at process time.
;
; See Also:     MSM\MSMDriverRemove
;
; END_MANUAL_ENTRY
;
;*********************************************************************/
;
DriverRemove    proc

        CPush
        mov     eax, DriverModuleHandle
        call    MSMDriverRemove
        CPop
        ret
;
DriverRemove    endp
;
OSCODE  ends

        end
```

```
/* interface between Helius DPCNE and Hughes DPCPE */

extern "C" {
#include <nwsemaph.h>
#include "sys_win.hhi"
#include "dpcutils.h"
#include "dbsinwin.h"
#undef VIRTUAL
#include "dpcagent.h"
#include <assert.h>

int PD_ESR(ECB*);
void DloHangup(void);
void DPCPDTerminate(void);
void DPCPDBackground(void);
void DPCFileMain(void* arg);     // thread
}
#include "sfxwatch.h"
#include "sfxqview.h"
#include "sfxparsr.h"

unsigned long GetTickCount(void) {
  return clock() * 1000 / CLOCKS_PER_SEC;
}

extern int DloState;
extern LONG DloPXmitCount;
extern LONG DloPMaxBufferSize;
extern LONG DloRcvCount;
extern LONG DloConn;

int DloGetCurrentState(void) {
#if 1
  return (DloState == DLOS_CONN && DloConn == DLO_CONN_PACKAGE) ? DLOS_CONN : DL
OS_IDLE;
#else
  return DloState;
#endif
}

int DloPortEmpty(void) {
#if 1
  return DloPXmitCount == 0;
#else
  return DloAndCommEmpty();
#endif
}

int DloPortOpen(void) {
  return AIOPortHandle != (-1);
}

int DloGetStatus(tDloStatus* pStatus) {
  if (pStatus == 0)
    return (-1);
  if (!DloPortOpen())
    return (-1);
  pStatus->iState = DloGetCurrentState();
  pStatus->iXmitBytesBuffered = DloPXmitCount;
  pStatus->iXmitBufferSpace = DloPMaxBufferSize;
  pStatus->iRcvBytesBuffered = DloRcvCount;
  pStatus->iRcvBufferSpace = DLOBUFSIZE;
  return 0;
}

int DloGetBufSize(void) {
```

```
  return DloPMaxBufferSize - DloPXmitCount;
}

DWORD DloExtendInactivityTimer(long) {
}

void DloHangup(void) {
}

void DloDispatch(void) {
}

/*
 * Returns whether the adapter can gain access to the passed group ID
 * The group ID includes a version number.
 */
long DLLAPI CDBCheckGroupID(CdbCfg_t *cfg)
{

        if(find_pacau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_IMPLICIT);
        if(find_dacau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_AUTHENTICATED);
        if(find_ecau(cfg->groupid, cfg->ver) != NULL)
                return(CAS_EXPLICIT);
        return(CAS_ERROR);
}

/*
 * Returns a version number which increments when there have been
 * ANY changes to the adapter's conditional access.
 */
long DLLAPI CDBCheckCAChange(void)
{
        return CDBVersion;
}


struct PID {
  PID()          { DPCFilePID = GetThreadID(); }
  ~PID()         { DPCFilePID = 0; }
};

struct Semaphore {
  LONG handle;
  Semaphore(long initial = 0) { handle = OpenLocalSemaphore(initial); }
  ~Semaphore() { if (handle) CloseLocalSemaphore(handle); }
  void Signal(void) { SignalLocalSemaphore(handle); }
  LONG Wait(int milliseconds = (-1)) { return TimedWaitOnLocalSemaphore(handle,
(LONG)milliseconds); }
  LONG value(void) { return ExamineLocalSemaphore(handle); }
  LONG operator --(void);
};

inline LONG Semaphore::operator --(void) {
  LONG v = value();
  if (v == 0)
    return v;
  WaitOnLocalSemaphore(handle);
  return v - 1;
}

Semaphore* DPCPDSemaphore;
SfxDispatcher* pDispatcher;
QUEUEVIEWER* pQueueViewer;
ECBQueue DPCPDQueue;
```

```
int PD_ESR(ECB* ecb) {
  Enqueue_IntsDisabled(&DPCPDQueue, ecb);
  return 0;
}

long BicddSignText(char* p_string,
                   unsigned long size,
                   char* p_sign) {
  return DIOSignText(p_string, size, p_sign);
}

long BicddGetSN(char* p_serial_num) {
  DIOGetSN(p_serial_num);
  return 0;
}

long BicddOpenChannel(BICDD_CHANNEL_CONFIG* channel_config) {
  if (channel_config->num_addresses != 1)
    return 1;

  DPCPDSemaphore = new Semaphore();
  if (DPCPDSemaphore->handle == 0) {
    delete DPCPDSemaphore;
    DPCPDSemaphore = 0;
    return 2;
  }
  DPCPDQueue.semaphore = DPCPDSemaphore->handle;

  // there is actually an overflow here IRT channel being a short!
  long ret = DIOOpenChannel(channel_config->address[0],
                            PD_ESR,
                            (LONG*)&channel_config->channel);
  if (ret) {
    delete DPCPDSemaphore;
    DPCPDSemaphore = 0;
    DPCPDQueue.semaphore = 0;
  }
  return ret;
}

long BicddCloseChannel(unsigned long channel) {
  long ret = DIOCloseChannel(channel);
  if (ret)
    return ret;
  delete DPCPDSemaphore;
  DPCPDSemaphore = 0;
  DPCPDQueue.semaphore = 0;
  while (DPCPDQueue.head) {
    ECB* ecb = Dequeue(&DPCPDQueue);
    CLSLReturnRcvECB(ecb);
  }
  return 0;
}

/***************************************************************************
 *                                                                         *
 *                          ELEMENTS SECTION                               *
 *                      ( Elements Table support )                         *
 *                                                                         *
 ***************************************************************************/

CDBelement_t Elements[MAXELEMENTS];

static find_element_by_mac(MACaddr_t mac)
{
```

```
  int k;
  int ret = -1;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
      ret = k;
      break;
    }
  }
  return ret;
}

static add_element(unsigned long channel, ID id, unsigned char ver,
                   MACaddr_t mac, char pack_feed)
{
  int k, ret = CAS_OK;

  if(find_element_by_mac(mac) != -1)
    return(CAS_DUPLICATE_ADDR);
  for(k = 0; k < MAXELEMENTS; k++)
    if(Elements[k].in_use != 'Y')
      break;
  if(k == MAXELEMENTS)
    ret = CAS_ERROR;
  else {
    Elements[k].channel = channel;
    Elements[k].e_ver = ver;
    memcpy(&Elements[k].e_id, &id, sizeof(id));
    memcpy(&Elements[k].e_mac, &mac, sizeof(mac));
    Elements[k].in_use = 'Y';
    Elements[k].packfeed = pack_feed;
  }
  return ret;
}

static find_element_id(ID id, unsigned char ver)
{
  int k;
  int ret = -1;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_id, &id, sizeof(id)) == 0 &&
       Elements[k].e_ver == ver) {
      ret = k;
      break;
    }
  }
  return ret;
}

static del_element_by_mac(MACaddr_t mac)
{
  int k, ret = CAS_ERROR;

  for(k = 0; k < MAXELEMENTS; k++) {
    if(Elements[k].in_use == 'Y' &&
       memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) {
      ret = CAS_OK;
      Elements[k].in_use = 'N';
      break;
    }
  }
  return ret;
}
```

```
/**************************************************************************
 * Add / Delete Package Delivery Address
 */

/*
 * Allows an application to request reseption of a single additional DPC MAC
 * address. Caller supplies the address's elementID and version number and the
 * element's group ID and version number. CDB looks up the group key and
 * element key for the address and attempts to add the address via a
 * driver call
 */
long BicddAddPKGAddr(CdbCfg_t* cfg) {
  char e_id_txt[7];
  MUXpacau_t* pacau;
  MUXdacau_t* dacau;

  make_element_id((BYTE*)&cfg->elementid, e_id_txt);
  MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);

  dacau = find_dacau(cfg->groupid, cfg->ver);
  pacau = dacau ? (MUXpacau_t*)dacau : find_pacau(cfg->groupid, cfg->ver);
  if(pacau == NULL)
    return CAS_ERROR;
  if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
    return CAS_ERROR;
  if (DIOAddGroupAddress(cfg->channel,
                         (BYTE*)&cfg->mac,
                         (BYTE*)&pacau->g_key) ==
      ESUCCESS)
    return CAS_OK;
  del_element_by_mac(cfg->mac);
  return CAS_ERROR;
}

/*
 * For use by package delivery. Allows an application to request
 * reception of a for-sale package ( a package from an explicit group).
 * Package delivery passes address to be received (including the version number)
 * plus the group key to be used to receive the package. This group key was
 * received via explicit request transaction with the NOC.
 * CDB creates the corresponding element key and calls WBicddAddress.
 */
long BicddAddExpAddr(CdbCfg_t* cfg) {
  if(find_ecau(cfg->groupid, cfg->ver) == 0)
    return CAS_ERROR;

  char e_id_txt[7];
  make_element_id((BYTE*)&cfg->elementid, e_id_txt);
  MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);
  if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
    return CAS_ERROR;
  if (DIOAddGroupAddress(cfg->channel,
                         (BYTE*)&cfg->mac,
                         (BYTE*)&cfg->expl_g_key) == ESUCCESS)
    return CAS_OK;
  del_element_by_mac(cfg->mac);
  return CAS_ERROR;
}


/*
 * Allows the application to discontinue reception of a single DPC MAC
 * Package Delivery supplies the element id and version number. CDB merely
 * reformats these values into a DPC MAC address and relays it to WINBICDD
 */
```

```
long BicddDeletePKGAddr(CdbCfg_t* cfg) {
  int element = find_element_id(cfg->elementid, cfg->ver);
  if (element == (-1))
    return CAS_ERROR;
  if (DIODeleteAddress(cfg->channel, (BYTE*)&Elements[element].e_mac))
    return CAS_ERROR;
  del_element_by_mac(Elements[element].e_mac);
  return CAS_OK;
}

long BicddPoll(unsigned long channel) {
  return DPCPDSemaphore ? DPCPDSemaphore->value() : (-1);
}

long BicddReceive(unsigned long channel,
                  BICDD_BUFFER* p_buffers,
                  unsigned long buf_size,
                  long timeout) {
  if (DPCPDSemaphore == 0)
    return (-1);
  if (DPCPDQueue.head == 0 && DPCPDSemaphore->Wait(timeout) != 0)
    return 0;
  ECB* ecb = DPCPDQueue.head;
  int r = 0;
  int n = buf_size / sizeof(BICDD_BUFFER);
  for (; ecb && n > 0; ecb = ecb->ECB_NextLink, --n) {
    p_buffers->data_size = ecb->ECB_Fragment[0].FragmentLength;
    p_buffers->buf_ptr = ecb->ECB_Fragment[0].FragmentAddress;
    p_buffers->last = 1;
    ++p_buffers;
    ++r;
  }
  return r * sizeof(BICDD_BUFFER);
}

long BicddFreeBuffers(unsigned long channel,
                      BICDD_BUFFER* p_buffers,
                      unsigned long buf_size) {
  int n = buf_size / sizeof(BICDD_BUFFER);
  int i = min(n, DPCPDSemaphore->value());
  for (; n > 0 && DPCPDQueue.head; --n)
    CLSLReturnRcvECB(Dequeue(&DPCPDQueue));
  for (; i > 0; --i)
    --*DPCPDSemaphore;
  return n;
}

long BicddGetSiteID(char* buffer) {
  if (!*SiteID)
    return (-1);
  strncpy(buffer, (char*)SiteID, 9);
  return 0;
}

BOOL BicddGetSatelliteStatus(BICDD_SAT_STATS* Stats, long chan) {
  Stats->MarginalCutoff = MARGINAL_ACQ_VALUE;
  Stats->NormalCutoff = NORMAL_ACQ_VALUE;
  Stats->CurrentValue = DPCGetSignalStrength();
  return TRUE;
}

// The name of the registry/ini key values accessed in this module

static char* pREGKEY_DeleteOnDelivery = "DeleteOnDelivery";
static char* pREGKEY_CooperativeLoading = "CooperativeLoading";
static char* pREGKEY_RebuildOnStartup = "RebuildOnStartup";
static char* pREGKEY_Reconcile = "Reconcile";
static char* pREGKEY_EnableDebug = "EnableDebug";
static char DBS_NAME[] = __FILE__;
static const char magic_key[] = {
 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11
};

/***************************************************************************
 *
 *       EXPORTED FUNCTION
 *
 *       DPCCancelDownload(LONG fileID)
 *
 *       Description:
 *               This routine cancels the download of the file associated
 *               with the fileID if one is pending. This means closing
 *               any open file handles and stoping the modem thread.
 *
 *       Input:
 *               fileID                          - File ID of file to cancel
 *
 *       Output:
 *               nothing
 *
 *       Returns:
 *               0               if download was canceled
 *
 ***************************************************************************/

static struct {
  LONG control;
  LONG ret;
  LONG fileID;
  BOOL cancel;
} crossover;


LONG DPCCancelDownload(LONG fileID)
{
  while (crossover.control)
    delay(100);
  crossover.fileID = fileID;
  crossover.cancel = TRUE;
  crossover.control = GetThreadID();
  while (crossover.control)
    delay(100);

  /* Force the help package status to idle */
  UpdateHelpPortal();

  return crossover.ret;
}

LONG DPCDownloadAFile(LONG fileID)
{
  while (crossover.control)
    delay(100);
  crossover.fileID = fileID;
  crossover.cancel = FALSE;
  crossover.control = GetThreadID();
  while (crossover.control)
    delay(100);

  return crossover.ret;
}
```

```
void DPCPDTerminate(void) {
  pDispatcher->Terminate();
}

void DPCPDBackground(void) {
  PDI_FillList_cross();

  if (crossover.control) {
    LONG fileID = crossover.fileID;
    if (fileID != fsm.getFileid() && fsm.unique(fileID) != SFX_OK)
      crossover.ret = (LONG)(-1);
    else if (crossover.cancel) {
      crossover.ret = fsm.dispatch(fileID, SFXFSM_FILE_NOT_WANTED);
      pDispatcher->CancelLoadingFileID(fileID);
    }
    else if (fsm.isRequestable(fileID)) {
      fsm.dispatch(fileID, SFXFSM_PRECOMMIT);
      crossover.ret =
        fsm.dispatch(fileID,
                     fsm.isForSale(fileID) ? SFXFSM_PURCHASE : SFXFSM_FILE_WANTE
D);
    }
  sendret:
    ResumeThread(crossover.control);
    crossover.control = 0;
  }
}


// this is adapted from sfxdemlp.cpp WinMain and dpcfile.c DPCFileMain

void DPCFileMain(void* arg) {
  PID pid;

  if (!PackageDelivery)
    return;

  DbsProcInit("DPCPD");

  if ((arg && stricmp((char*)arg, "rebuild") == ESUCCESS) ||
      DPCGetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_RebuildOnStartup, 0)) {
    DPCSetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_RebuildOnStartup, 0);

    int n = fsm.Rebuild();
    if (n >= 0) {
      DBS_SEND_TRACE1(0, "File database rebuilt with %d entries restored", n);
    }
    else {
      DBS_SEND_TRACE("File database rebuild failed");
    }

    return;
  }

  // wait until DIOBoard initialized
  while (DIOBoard == 0) {
    if (ExitingFlag)
      return;
    delay(500);
  }

  pDispatcherLro = new SfxDispatcherLro();
  if (!pDispatcherLro) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcherLro");
    goto cleanup;
  }
  pDispatcher = new SfxDispatcher();
  if (!pDispatcher) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcher");
    goto cleanup;
  }
  pQueueViewer = new QUEUEVIEWER(PDI_UpdateDisplay);
  if (!pQueueViewer) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct QUEUEVIEWER");
    goto cleanup;
  }

  if (DPCGetProfileInt(PROF_PACKAGEDELIVERY, pREGKEY_Reconcile, 1))
    fsm.ReconcileWith(frd);
  cdb.rebuildDB();

  while (!ExitingFlag) {
    pDispatcher->Run();
    DPCPDBackground();
  }

  pDispatcher->Stop(5000);

  cleanup:
  delete pQueueViewer;
  delete pDispatcher;
  delete pDispatcherLro;
}
```

```
#include "dpcagent.h"           /* Our header file */

/* "fix" conflicting types */
#define AllocateResourceTag __AllocateResourceTag__
#include <advanced.h>
#undef AllocateResourceTag
#include <nwbitops.h>

#define milliclock()     (GetHighResolutionTimer() / 10)

#define activityTimer   ECB_DriverWorkspace.DWs_i32val

/* various flags that control the filter */
#undef  FILTER_DATA_ON_RST
#define WIDEN_TCP_WINDOW
#undef  TCP_ACK_LATENCY /* 10 */
/* if used at all, define *only* 1 of the following */
#undef  DPCInetMaxQueuedBytes   /* 4096 */
#define DPCInetMaxQueuedPackets 64
#if     defined(DPCInetMaxQueuedBytes) && defined(DPCInetMaxQueuedPackets)
#error  Only 1 of DPCInetMaxQueuedBytes and DPCInetMaxQueuedPackets allowed
#endif

/* various flags that control the tunnel */
#undef  TUNNEL_ONLY_TCP

#define IP_VERS(x)              (((BYTE*)x)[0] >> 4)
#define IP_HD_LEN(x)            (((BYTE*)x)[0] & 0x0f)
#define IP_TOS(x)               (((BYTE*)x)[1])
#define IP_TOT_LEN(x)           (((WORD*)x)[1])
#define IP_FLAG_FRAG(x)         (((WORD*)x)[3])
#define IP_PROTO(x)             (((BYTE*)x)[9])
#define IP_CSUM(x)              (((WORD*)x)[5])
#define IP_SRC_ADDR(x)          (((LONG*)x)[3])
#define IP_DST_ADDR(x)          (((LONG*)x)[4])

#define IPPROTO_IPENCAP         0x04

#define UDP_SRC_PORT(x)         (((WORD*)x)[0])
#define UDP_DST_PORT(x)         (((WORD*)x)[1])

#define TCP_SRC_PORT(x)         (((WORD*)x)[0])
#define TCP_DST_PORT(x)         (((WORD*)x)[1])
#define TCP_ACKNUM(x)           (((LONG*)x)[2])
#define TCP_CODE(x)             (((BYTE*)x)[13])
#define TCP_WINDOW(x)           (((WORD*)x)[7])
#define TCP_CSUM(x)             (((WORD*)x)[8])

#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20

ECBQueue TxQ;
ECBQueue NewQ;

struct ResourceTagStructure* TxChainRTag = 0;
struct ResourceTagStructure* TxECBRTag = 0;
LONG TxChainID;
struct ResourceTagStructure* RxChainRTag = 0;
struct ResourceTagStructure* RxECBRTag = 0;
LONG RxChainID;
LONG DPC_IP_Address = 0;
static BYTE ConnectionMask[65536 / 8];
```

```

#ifndef __GNUC__
#define inline
#endif  /* __GNUC__ */


/* ECB Manipulation */

static inline void ReleaseECB(ECB* ecb) {
  if (DIOStats) {
#ifdef  DPCInetMaxQueuedBytes
  if ((DIOStats->QDepth -= ecb->ECB_DataLength) < 0)
    DIOStats->QDepth = 0;
#endif
#ifdef  DPCInetMaxQueuedPackets
    --DIOStats->QDepth;
#endif
  }
  --TxECBRTag->RTResourceCount;
  CLSLFastSendComplete(ecb);
#ifdef LOG_ECB_ACTIVITY
  FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                           "TINET Release(%08lx)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
}


inline void Enqueue_IntsDisabled(ECBQueue* q, ECB* ecb) {
  ecb->ECB_NextLink = 0;
  if (q->tail)
    q->tail->ECB_NextLink = ecb;
  ecb->ECB_PreviousLink = q->tail;
  q->tail = ecb;
  if (q->head == 0)
    q->head = ecb;
  SignalLocalSemaphore(q->semaphore);
}


void Enqueue(ECBQueue* q, ECB* ecb) {
  _disable();
  Enqueue_IntsDisabled(q, ecb);
  _enable();
}


ECB* Dequeue(ECBQueue* q) {
  ECB* ecb;
  _disable();
  ecb = q->head;
  if (ecb == 0) {
    _enable();
    return 0;
  }
  q->head = ecb->ECB_NextLink;
  if (q->head == 0)
    q->tail = 0;
  else
    q->head->ECB_PreviousLink = 0;
  _enable();
  ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
  return ecb;
}
```

```
void Remove(ECBQueue* q, ECB* ecb) {
  _disable();
  if (ecb->ECB_NextLink)
    ecb->ECB_NextLink->ECB_PreviousLink = ecb->ECB_PreviousLink;
  else
    q->tail = ecb->ECB_PreviousLink;
  if (ecb->ECB_PreviousLink)
    ecb->ECB_PreviousLink->ECB_NextLink = ecb->ECB_NextLink;
  else
    q->head = ecb->ECB_NextLink;
  _enable();
  ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
}


LONG InetQueuePacket(ECB* ecb, LONG board, void* chainID) {
  board = board;                        /* not used */
  chainID = chainID;                    /* not used */

  /* only handle IP packets */
  if (*(LONG*)ecb->ECB_ProtocolID != 0 ||
      *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

#ifdef LOG_ECB_ACTIVITY
  if (LogECBHandle) {
    int TGID = SetThreadGroupID(DPC_TGID);
    LogMsg(LogClientHandle, LogECBHandle, FALSE,
           "TINET Enqueue(%08lx)\n", ecb);
    SetThreadGroupID(TGID);
  }
#endif /* LOG_ECB_ACTIVITY */
  Enqueue(&NewQ, ecb);
  return 0;
}


LONG InetControl(void) {
  return 0xffffff81;
}


void ClearConnection(WORD port) {
  if (ScanBits(ConnectionMask, port, port+2) == port) {
    BitClear(ConnectionMask, port);
    --DIOStats->TxOKMultipleCollisions;
  }
}


int AllocateConnection(WORD port) {
  if (ScanBits(ConnectionMask, port, port+2) != port) {
    /* see if there is a connection left */
    if (DIOStats->TxOKMultipleCollisions < DPCMaxConnections) {
      /* allocate the new connection */
      BitSet(ConnectionMask, port);
      ++DIOStats->TxOKMultipleCollisions;
      return 1;
    }
    return 0;
  }
  return 1;
}


LONG ConnectionLimiter(ECB* ecb, LONG board, void* chainID) {
```

```
  BYTE* IPHeader = ecb->ECB_Fragment[0].FragmentAddress;
  BYTE* TCPHeader = 0;
  board = board;                        /* not used */
  chainID = chainID;                    /* not used */

  /* only handle IP packets */
  if (*(LONG*)ecb->ECB_ProtocolID != 0 ||
      *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

  /* double check stats, hopefully upper layer is kosher, but */
  if (DIOStats == 0) {
  releaseECB:
    --RxECBRTag->RTResourceCount;
    CLSLFastSendComplete(ecb);
    return 0;
  }

  /* only check TCP packets to our interface */
  if (IP_PROTO(IPHeader) != IPPROTO_TCP ||
      IP_DST_ADDR(IPHeader) != DPC_IP_Address)
    return 1;

  TCPHeader = IPHeader + IP_HD_LEN(IPHeader) * 4;

  if (ecb->ECB_Fragment[0].FragmentLength < ((TCPHeader + 20) - IPHeader))
    return 1;

  if (TCP_CODE(TCPHeader) & (TCP_FIN|TCP_RST)) {
    /* release the connection */
    ClearConnection(ntohs(TCP_DST_PORT(TCPHeader)));
  }
  else if (TCP_CODE(TCPHeader) & TCP_SYN) {
    /* allocate the connection */
    if (!AllocateConnection(ntohs(TCP_DST_PORT(TCPHeader))))
      goto releaseECB;
  }

  return 1;
}


/* IP Manipulation */

#if 0
char *chksum (BYTE *buf, unsigned cnt)
{
        static unsigned char crc_bytes[2];
        BYTE rbl;
        WORD rax, rcx;
        int redx;
        BYTE *rdssi;

        crc_bytes[0] = crc_bytes[1] = 0;

        rcx = cnt;
        rdssi = buf;
        rbl = rcx;
        rcx = rcx >> 1;
        redx = 0;
        if (rcx != 0)
        {
                while (rcx--)
                {
                        rax = *((WORD *)rdssi);
                        rdssi += 2;
                        if (redx & 0xffff0000)
                                redx++;
                        redx &= 0x0000ffff;
                        redx += rax;
                }
                if (redx &= 0x0000ffff)
                {
                        redx &= 0x0000ffff;
                        redx++;
                }
        }
        if (rbl & 1)
        {
                rax = 0;
                rax = *rdssi;
                redx += rax;
                if (redx &= 0x0000ffff)
                        redx++;
        }
        redx = ~redx;
        crc_bytes[0] = redx & 0xff;
        crc_bytes[1] = (redx >> 8) & 0xff;
        return (char *)crc_bytes;
};

#endif


#ifdef  __GNUC__
/*
 *      This is a version of ip_compute_csum() optimized for IP headers, which
 *      always checksum on 4 octet boundaries.
 *      this version is constructed from various places in the linux and Hughes
 *      sources.
 */

static inline unsigned short ip_fold_1comp_csum(unsigned long sum) {
  unsigned short csum;
  __asm__("movl %w1, %w0\n\t"
          "shrl $16, %1\n\t"
          "addw %w1, %w0\n\t"
          "adcw $0, %w0\n\t"
          "notw %w0"
          : "=a" (csum)
          :  "b" (sum));
  return csum;
}

static inline unsigned short ip_fast_csum(unsigned short * buff, int wlen) {
  unsigned long sum = 0;

  if (wlen) {
    unsigned long eax;
    /* Suggested speedup:
       1:
       movl (%esi), %ebx
       lea (%esi+4), %esi
       adcl %ebx, %eax
       decl %ecx
        jnz 1b
       adcl $0, %eax
       movl %eax, %ebx
       shrl $16, %eax
       addw %ebx, %eax
       adcl $0, %eax
```

```
       xorl $0xffff, %eax
       */
    __asm__("clc\n"
            "1:\t"
            "lodsl\n\t"
            "adcl %3, %0\n\t"
            "loop 1b\n\t"
            "adcl $0, %0\n\t"
            : "=r" (sum), "=S" (buff), "=c" (wlen), "=a" (eax)
            :  "0" (sum),  "1" (buff),  "2" (wlen));
  }
  return ip_fold_1comp_csum(sum);
}

#define chksum(b, l)    ip_fast_csum(b, (l) / 4)

static inline unsigned short ip_adjust_csum(unsigned short oldcsum,
                                            unsigned short oldval,
                                            unsigned short newval) {
  unsigned long sum = ((unsigned short)~oldcsum);
  sum += ((unsigned short)~oldval);
  sum += newval;
  return ip_fold_1comp_csum(sum);
}

#endif /* __GNUC__ */


static int DummyFrame(FRAG_DESC* frag) {
  frag = frag;                                /* not used */
  return 0;
}


int (*DPCDropFrame)(FRAG_DESC* frag) = DummyFrame;


void FilterQueue(void* arg) {
  ECB* ecb;
  ECB* rover;
  BYTE* IP;
  BYTE* TCP;
  int excess;
  arg = arg;                                  /* not used */

  RenameThread(GetThreadID(), "DPCAgent Filter");

  for (;;) {
    if (ExitingFlag)
      return;
    TimedWaitOnLocalSemaphore(NewQ.semaphore, 1000);
    if (!NewQ.head)
      continue;

    ecb = Dequeue(&NewQ);
    ecb->activityTimer = milliclock();
    IP = ecb->ECB_Fragment[0].FragmentAddress;

    if (DIOStats == 0) {
    releaseECB:
      DPCDropFrame((FRAG_DESC*)&ecb->ECB_FragmentCount);
      --TxECBRTag->RTResourceCount;
      CLSLFastSendComplete(ecb);
#ifdef LOG_ECB_ACTIVITY
      FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
                                "TINET Release(%08lx)\n", ecb));
```

```
#endif /* LOG_ECB_ACTIVITY */
      continue;
    }

    /* always send a fragmented or routed packet */
    if (ecb->ECB_Fragment[0].FragmentLength < 20 ||
        IP_FLAG_FRAG(IP) & htons(0x3fff) ||
        IP_SRC_ADDR(IP) != DPC_IP_Address) {
    enqueueTxQ:
#ifdef  DPCInetMaxQueuedBytes
      if (DIOStats->QDepth > DPCInetMaxQueuedBytes) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
      }
      DIOStats->QDepth += ecb->ECB_DataLength;
#endif
#ifdef  DPCInetMaxQueuedPackets
      if (DIOStats->QDepth > DPCInetMaxQueuedPackets) {
        ++DIOStats->RetryTxCount;
        goto releaseECB;
      }
      ++DIOStats->QDepth;
#endif
      Enqueue(&TxQ, ecb);
      continue;
    }

    excess = ecb->ECB_Fragment[0].FragmentLength - IP_HD_LEN(IP) * 4;
    if (excess > 0) {
      TCP = IP + IP_HD_LEN(IP) * 4;
    }
    else {
      TCP = ecb->ECB_Fragment[1].FragmentAddress + (-excess);
      excess += ecb->ECB_Fragment[1].FragmentLength;
    }

    if (IP_PROTO(IP) == IPPROTO_UDP)
      goto filterUDP;

    if (IP_PROTO(IP) != IPPROTO_TCP)
      goto enqueueTxQ;

  filterTCP:
    if (excess < 20)
      goto enqueueTxQ;

    if (TCP_CODE(TCP) & TCP_SYN) {
      if (!AllocateConnection(ntohs(TCP_SRC_PORT(TCP))))
        goto releaseECB;
      /* scan for duplicate in TxQ */
      for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverTCP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
                  IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
          roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
          roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
          excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 20 &&
            (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_TCP &&
```

```
            TCP_CODE(roverTCP) == TCP_CODE(TCP) &&
            IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
            TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
            IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
            TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP)) {
          ++DIOStats->TxOKSingleCollision;
          goto releaseECB;
        }
      }
      goto enqueueTxQ;
    }
#ifdef  FILTER_DATA_ON_RST
    if (TCP_CODE(TCP) & TCP_RST) {
      /* scan for data in TxQ */
      for (rover = TxQ; rover; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverTCP;
        excess = (rover->ECB_Fragment[0].FragmentLength -
                  IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
          roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
        }
        else {
          roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
          excess += rover->ECB_Fragment[1].FragmentLength;
        }
        if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
            excess >= 20 &&
            (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
            IP_PROTO(roverIP) == IPPROTO_TCP &&
            IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
            TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
            IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
            TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
            TCP_CODE(roverTCP) != TCP_CODE(TCP)) {
          rover->activityTimer = 0;     /* will get taken out shortly */
          ++DIOStats->TxAbortCarrierSense;
        }
      }
      /* fallthru */
    }
#endif
    if (TCP_CODE(TCP) & (TCP_RST|TCP_FIN)) {
      ClearConnection(ntohs(TCP_SRC_PORT(TCP)));
      goto enqueueTxQ;
    }
    if (TCP_CODE(TCP) & TCP_ACK) {
#ifdef WIDEN_TCP_WINDOW
      WORD oldwin = TCP_WINDOW(TCP);
      WORD newwin = ntohs(oldwin);
      if (newwin < 40000) {
        newwin += (newwin >> 1);          /* *= 1.5, will be <= 60K' */
        newwin = htons(newwin);
        TCP_WINDOW(TCP) = newwin;
        TCP_CSUM(TCP) = ip_adjust_csum(TCP_CSUM(TCP),
                                       oldwin,
                                       newwin);
      }
#endif /* WIDEN_TCP_WINDOW */
      if (TCP_CODE(TCP) & (TCP_URG|TCP_PSH))
        goto enqueueTxQ;
#ifdef TCP_ACK_LATENCY
      ecb->activityTimer = milliclock() + TCP_ACK_LATENCY;
#endif
      /* scan for redundancy in TxQ */
      for (rover = TxQ head; rover; rover = rover->ECB_NextLink) {
```

```
    BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
    BYTE* roverTCP;
    excess = (rover->ECB_Fragment[0].FragmentLength -
              IP_HD_LEN(roverIP) * 4);
    if (excess > 0) {
      roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
    }
    else {
      roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
      excess += rover->ECB_Fragment[1].FragmentLength;
    }
    if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
        excess >= 20 &&
        (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
        IP_PROTO(roverIP) == IPPROTO_TCP &&
        IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
        TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
        IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
        TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
        TCP_CODE(roverTCP) & TCP_ACK &&
        (htonl(TCP_ACKNUM(roverTCP)) + htons(TCP_WINDOW(roverTCP)) <
         htonl(TCP_ACKNUM(TCP)) + htons(TCP_WINDOW(TCP)))) {
      /* move ACK information over to TxQ and release this packet */
      TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                          TCP_WINDOW(roverTCP),
                                          TCP_WINDOW(TCP));
      TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                          (WORD)TCP_ACKNUM(roverTCP),
                                          (WORD)TCP_ACKNUM(TCP));
      TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
                                          TCP_ACKNUM(roverTCP)>>16,
                                          TCP_ACKNUM(TCP)>>16);
      TCP_ACKNUM(roverTCP) = TCP_ACKNUM(TCP);
      TCP_WINDOW(roverTCP) = TCP_WINDOW(TCP);
      ++DIOStats->TxAbortExcessCollisions;
      goto releaseECB;
    }
  }
  goto enqueueTxQ;
}
goto enqueueTxQ;

filterUDP:
  {
    BYTE* UDP = TCP;
    BYTE* DNS;

    /* ECB contents determined by inspection, there are safer methods */
    if (excess < 8)
      goto enqueueTxQ;

    /* filter DNS only */
    if (UDP_DST_PORT(UDP) != htons(53))
      goto enqueueTxQ;

    excess -= 8;
    DNS = (excess > 0) ? (UDP + 8) : ecb->ECB_Fragment[1].FragmentAddress;

    for (rover = TxQ.head; rover; rover = rover->ECB_NextLink) {
      BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
      BYTE* roverUDP;
      BYTE* roverDNS;
      excess = (rover->ECB_Fragment[0].FragmentLength -
                IP_HD_LEN(roverIP) * 4);
      if (excess > 0) {
        roverUDP = roverIP + IP_HD_LEN(roverIP) * 4;
```

```
      }
      else {
        roverUDP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
        excess += rover->ECB_Fragment[1].FragmentLength;
      }
      if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
          excess >= 8 &&
          (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
          IP_PROTO(roverIP) == IPPROTO_UDP &&
          IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
          UDP_DST_PORT(roverUDP) == UDP_DST_PORT(UDP) &&
          IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
          UDP_SRC_PORT(roverUDP) == UDP_SRC_PORT(UDP) &&
          (roverDNS = (((excess -= 8) > 0) ?
                       (roverUDP + 8) :
                       rover->ECB_Fragment[1].FragmentAddress)) &&
          *(LONG*)DNS == *(LONG*)roverDNS) {
        ++DIOStats->TxAbortLateCollision;
        goto releaseECB;
      }
    }
    goto enqueueTxQ;
  }
 }
}


/* SLIP, PPP, Modem Manipulation */

#define MAX_READ_BUF 128

int InetState = MODEM_IDLE;
static BYTE SlipEndPkt[1] = {END};


int WaitingLines = 0, NextWait = 0;
char WaitingBuffer[MAX_READ_BUF];
int WaitingIndex = 0;
LONG ConnectingTimeout = 0;
LONG ConnectingRedial = FALSE;

int BaudRate[] =
{
        2400,                               /* 0 */
        3600,                               /* 1 */
        4800,                               /* 2 */
        7200,                               /* 3 */
        9600,                               /* 4 */
        19200,                      /* 5 */
        38400,                      /* 6 */
        57600,                      /* 7 */
        115200                      /* 8 */
};


void InitLogin()
{
        int i;
        char *nextWait;

        WaitingLines = 0;
        if (DloCfg.auto_login)
        {
                WaitingIndex = 0;
                WaitingBuffer[WaitingIndex] = '\0';
                NextWait = 0;
                ConnectingTimeout = 0;
                for (i = 0, nextWait = DloCfg.wait_for_1; i < 9 ; i++, nextWait+
=sizeof(DloCfg.wait_for_1))
                {
                        if (*nextWait)
                                WaitingLines++;
                }
        }

}


static BYTE MTUBuffer[8192];


int SLIPSendRoutineOpt(FRAG_DESC* fragStruc)
{
        LONG count = 0;
        BYTE* output = MTUBuffer;

        *output++ = END;
        while (count < fragStruo->FragmentCount)
        {
                FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
                BYTE* frame = (BYTE*)frag->FragmentAddress;
                LONG length = frag->FragmentLength;

                while (length-- > 0)
                {
                        switch (*frame)
                        {
                                case END:
                                        *output++ = ESC;
                                        *output++ = ESC_END;
                                        break;
                                case ESC:
                                        *output++ = ESC;
                                        *output++ = ESC_ESC;
                                        break;
                                default:
                                        *output++ = *frame;
                                        break;
                        }
                        ++frame;
                }
                ++count;
        }
        *output++ = END;

        if (output - MTUBuffer < 22 ||
            output - MTUBuffer > DloGetWriteBufferSpace())
                return 0;               /* oh, well */
        DloSend(MTUBuffer, output - MTUBuffer, DLO_INET_TIMEOUT);
        return 1;
}


int SLIPSendRoutineDebug(FRAG_DESC* fragStruc)
{
        LONG count = 0;
        BYTE* output = MTUBuffer;
        BYTE* dataStart = 0;            /* separate HEADER from PAYLOAD */
        LONG header = 0x80000000;

        while (count < fragStruc->FragmentCount)
        {
```

```
                FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
                BYTE* frame = (BYTE*)frag->FragmentAddress;
                LONG length = frag->FragmentLength;

                while (length-- > 0)
                {
                        switch (*frame)
                        {
                                case END:
                                        *output++ = ESC;
                                        *output++ = ESC_END;
                                        break;
                                case ESC:
                                        *output++ = ESC;
                                        *output++ = ESC_ESC;
                                        break;
                                default:
                                        *output++ = *frame;
                                        break;
                        }
                        if (header == 0x80000000)
                                header = (*frame & 0x0f) * 4;
                        if (--header == 0)
                                dataStart = output;
                        ++frame;
                }
                ++count;
        }

        if (output - MTUBuffer < 20 ||
            output - MTUBuffer > DloGetWriteBufferSpace())
                return 0;               /* oh, well */
        DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
        DloSend(MTUBuffer, dataStart - MTUBuffer, DLO_INET_TIMEOUT);
        DloSend(dataStart, output - dataStart, DLO_INET_TIMEOUT);
        DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
        return 1;
}

int (*DPCTxFrame)(FRAG_DESC* fragStruc) = SLIPSendRoutineOpt;

/***************************************************************************
*
*       IPSendRoutine(ECB *tcb)
*
*       Description:
*
*       Input:
*               ecb                                             - Pointer to Eve
nt Control Block
*
*       Output:
*               nothing
*
*       Returns:
*               0 if finished with ECB
*
***************************************************************************/

static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
        0x45,                           /* version 4, length 5 */
        0,                              /* tos */
        0, 0,                           /* length */
        0, 0,                           /* ident */
        0, 0,                           /* fragment */
        0x7f,                           /* ttl */
        4,                              /* IP in IP (encapsulation) */
};
#define IPHeaderIdent   (*(WORD*)&IPHeader[4])

int     IPSendRoutine(ECB *ecb)
{
        FRAG_DESC* fragStruc = alloca(sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * (
ecb->ECB_FragmentCount + 2)));
        WORD frame_size = ecb->ECB_DataLength;
        int options_collapsed = 1;
        LONG currFrag = 0;
        BYTE* ecbIPHeader = ecb->ECB_Fragment[0].FragmentAddress;

        /* initialize the copy of the tcb fragStruct */
        memcpy(fragStruc,
               &ecb->ECB_FragmentCount,
               sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * ecb->ECB_FragmentCount))
;
        if (frame_size < DloCfg.mtu &&
            (
#ifdef  TUNNEL_ONLY_TCP
             /* UDP doesn't need tunnel header */
             (ecbIPHeader[9] != IPPROTO_TCP) ||
#endif
             /* either do "routed" packets */
             (*(LONG*)&ecbIPHeader[12] != DPC_IP_Address)))
                goto skipFragger;

        memset(&fragStruc->FragmentDesc[fragStruc->FragmentCount],
               0,
               sizeof(FRAGMENTSTRUCT) * 2);

        frame_size += IP_TUNNEL_SIZE;
        /*
         * fill IPHeader with tunnel data, including IP/gateway addresses,
         * and prepend to frag list.
         */
        *(WORD*)(&IPHeader[2]) = htons(frame_size);
        ++IPHeaderIdent;
        *(WORD*)(&IPHeader[10]) = 0; /* checksum, for now */
        *(LONG*)(&IPHeader[12]) = DloCfg.ip_address;
        *(LONG*)(&IPHeader[16]) = DloCfg.gateway_address;
        memmove(fragStruc->FragmentDesc + 1,
                fragStruc->FragmentDesc,
                sizeof(FRAGMENTSTRUCT) * fragStruc->FragmentCount);
        fragStruc->FragmentDesc[0].FragmentAddress = IPHeader;
        fragStruc->FragmentDesc[0].FragmentLength = IP_TUNNEL_SIZE;
        ++fragStruc->FragmentCount;
        ++currFrag;
        *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader,
                                         IP_TUNNEL_SIZE);

        while (frame_size > DloCfg.mtu)
        {
                /*
                 * Shucks.  Have to fragment the packet.
                 * This algorithm is roughly per RFC791.
                 */
                LONG OIHL = fragStruc->FragmentDesc[0].FragmentLength;
                BYTE OMF = IPHeader[6] & 0x20;
                LONG NFB = (DloCfg.mtu - OIHL) & 0xfff8;
                WORD TL = OIHL + NFB;
```

```
                IPHeader[6] |= 0x20;    /* set More Fragments */
                *(WORD*)(&IPHeader[2]) = htons(TL);
                *(WORD*)(&IPHeader[10]) = 0; /* clear checksum */
                *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader, OIHL);

                /*
                 * Now fake out the fragStruc to reflect TL.
                 * Hang on to enough information to remove the TL less OIHL
                 * later.
                 */
                TL -= OIHL;
                frame_size -= TL;
                while (TL > 0 &&
                       fragStruc->FragmentDesc[currFrag].FragmentLength <= TL)
                {
                        TL -= fragStruc->FragmentDesc[currFrag].FragmentLength;
                        ++currFrag;
                }
                if (TL > 0)
                {
                        /*
                         * This frag gets split into 2 pieces.
                         */
                        memmove(fragStruc->FragmentDesc + currFrag + 1,
                                fragStruc->FragmentDesc + currFrag,
                                sizeof(FRAGMENTSTRUCT) *
                                (fragStruc->FragmentCount - currFrag));
                        ++fragStruc->FragmentCount;
                        fragStruc->FragmentDesc[currFrag].FragmentLength = TL;
                        ++currFrag;
                        fragStruc->FragmentDesc[currFrag].FragmentLength -= TL;
                        fragStruc->FragmentDesc[currFrag].FragmentAddress = ((ch
ar*)fragStruc->FragmentDesc[currFrag].FragmentAddress) + TL;
                }
                TL = fragStruc->FragmentCount - currFrag + 1;
                fragStruc->FragmentCount = currFrag;

                if (DPCTxFrame(fragStruc) == 0)
                        return 0;

                if (!options_collapsed) {
                        LONG offset = 20;
                        while (offset < OIHL && IPHeader[offset]) {
                                if (IPHeader[offset] & 0x80) /* copy */
                                        offset += IPHeader[offset + 1];
                                else {  /* collapse */
                                        LONG len = IPHeader[offset + 1];
                                        memcpy(IPHeader + offset,
                                               IPHeader + offset + len,
                                               OIHL - (offset + len));
                                        OIHL -= len;
                                }
                        }
                        offset = fragStruc->FragmentDesc[0].FragmentLength;
                        fragStruc->FragmentDesc[0].FragmentLength = (OIHL + 3) &
 0x3c;
                        memset(IPHeader + OIHL,
                               0,
                               fragStruc->FragmentDesc[0].FragmentLength - OIHL)
;
                        IPHeader[0] = 0x40 | (fragStruc->FragmentDesc[0].Fragmen
tLength / 4);
                        frame_size -= offset - fragStruc->FragmentDesc[0].Fragme
ntLength;
                        options_collapsed = 1;
                }
                /*
                 * Adjust the frag list to "remove" the fragment just sent.
                 */
                memmove(fragStruc->FragmentDesc + 1,
                        fragStruc->FragmentDesc + currFrag,
                        sizeof(FRAGMENTSTRUCT) * TL);
                fragStruc->FragmentCount = TL;
                currFrag = 1;

                /* compute new IPHeader values: fragment offset */
                *(WORD*)(&IPHeader[6]) = htons((ntohs(*(WORD*)(&IPHeader[6])) &
                                                0x1fff)
                                              + (NFB / 8));
                IPHeader[6] |= OMF;
                if (frame_size <= DloCfg.mtu)
                {
                        *(WORD*)(&IPHeader[2]) = htons(frame_size);
                        *(WORD*)(&IPHeader[10]) = 0; /* clear checksum */
                        *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader,
                                                        fragStruc->FragmentDesc
[0].FragmentLength);
                        break;
                }
        }

skipFragger:
        /* send the remaining (possibly ALL) portion of the frame */
        if (DPCTxFrame(fragStruc) == 0)
                return 0;

        if (DIOStats)
        {
                ++DIOStats->TotalTxPacketCount;
                if ((DIOStats->TotalTxOKByteCountLow += ecb->ECB_DataLength) <
                    ecb->ECB_DataLength)
                        ++DIOStats->TotalTxOKByteCountHigh; /* wrapped */
        }
        return 1;
}


static void EmptyESR(ECB* ecb) {
}

unsigned char RawEnvelope[14] = {
        0x00, 0x00, 0x0c, 0x0a, 0x0b, 0x0c,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x08, 0x00,
};

ECB RawECB = {
        0,                              /* ECB_NextLink */
        0,                              /* ECB_PreviousLink */
        0,                              /* ECB_Status */
        EmptyESR,                       /* ECB_ESR */
        -1,                             /* ECB_StackID */
        "",                             /* ECB_ProtocolID */
        0,                              /* ECB_BoardNumber */
        "",                             /* ECB_ImmediateAddress */
        { 0 },                          /* ECB_DriverWorkspace */
        { 0 },                          /* ECB_ProtocolWorkspace */
        -1,                             /* ECB_DataLength */
        -1,                             /* ECB_FragmentCount */
        { RawEnvelope, sizeof(RawEnvelope) },
};
```

```
FRAGMENTSTRUCT RawFrags[15] = {         /* technically unsafe, but works */
        IPHeader, sizeof(IPHeader),
};

#define RawSendRoutineDebug RawSendRoutineOpt
int RawSendRoutineOpt(FRAG_DESC* fragStruc) {
  int i;
  for (i = 20000; --i > 0; ) {
    if (RawECB.ECB_Status == 0)
      goto rawSend;
    if (ExitingFlag)
      return 0;
    ThreadSwitchWithDelay();
  }
  return 0;
rawSend:
  i = fragStruc->FragmentCount;
  RawECB.ECB_FragmentCount = i + 1;
  memcpy(RawFrags,
         fragStruc->FragmentDesc,
         i * sizeof(FRAGMENTSTRUCT));
  RawECB.ECB_DataLength = sizeof(RawEnvelope);
  while (i > 0)
    RawECB.ECB_DataLength += RawFrags[--i].FragmentLength;
  CLSLSendPacket(&RawECB);
  return 1;
}

void DisplayWaitStatus(void)
{
        char statusStr[80];
        char *waitingStr;

        if (WaitingLines == 0)
                return;

        waitingStr = (char *)&DloCfg.wait_for_1[NextWait*30];
        NWsprintf(statusStr, MSG("Modem Status: Waiting for \"%s\"\n", 557), wai
tingStr);
        UpdateModemStr(statusStr);
}

void InetStateChange(int state) {
  if (DloCfg.out_protocol == OUT_NETWORK) {
    InetState = PROTOCOL_CONNECTED;
    return;
  }
  switch (state) {
  case DLOS_IDLE:
  case DLOS_DISC_1:
  case DLOS_DISC_2:
  case DLOS_DISC_3:
  case DLOS_DISC_4:
    InetState = MODEM_IDLE;
    if (ConnectingRedial) {
      DloEndConn();
      ConnectingRedial = FALSE;
    }
    break;

  case DLOS_CONN:
    InetState = MODEM_CONNECTED;
    InitLogin();
    if (InetAsleep)
      ResumeThread(DPCInetPID);
    break;
```

```
 default:
   break;
 }

}

/*************************************************************************
 * FUNCTION: Convert Internet addess Address
 *
 * DESCRIPTION: converts a character string containing the Internet address
 * into a form that BIC DD understands.
 * e.g. 139.85.124.06 (8B.55.7C.06)  into 067C558B0000
 ************************************************************************/
void convert_address(char *lpszIpAddress)
{
        char *p;
        int i = 0;
        char tmp[20],tmp1[10];
        tmp[0] = 0;
        while((p=strrchr(lpszIpAddress,(int)'.')) != NULL)
        {
                i = atoi(p+1);
                NWsprintf(tmp1,MSG("%02X", 477),i);
                strcat(tmp,tmp1);
                *p = 0;
        }
        i = atoi(lpszIpAddress);
        NWsprintf(tmp1,MSG("%02X", 478),i);
        strcat(tmp,tmp1);
        strcat(tmp,MSG("0000", 479));
        CStrCpy(lpszIpAddress, tmp);
}

MACaddr_t       HIAddr;
LONG    InetChannel;

void make_hi_key(chunk *key)
{
        int i;
        LONG sn;
        BYTE serialNum[9];
        BYTE serialNumPacked[3];
        BYTE x;

        DIOGetSN(serialNum);
        sn = atol(serialNum);
        NWsprintf(serialNum, MSG("%061x", 480), sn);

        pack_mac_addr(serialNumPacked, 3, serialNum, 6);
        x = serialNumPacked[0];
        serialNumPacked[0] = serialNumPacked[2];
        serialNumPacked[2] = x;

  key->b[0] = serialNumPacked[0] ^ 0xff;
  key->b[1] = serialNumPacked[1] ^ 0xff;
  key->b[2] = serialNumPacked[2] ^ 0xff;
  for(i = 3; i < 8; i++)
        key->b[i] = 0x00 ^ 0xff;

        MACbuildAddr(serialNum, MAC_HI, 0, &HIAddr);
}

void InetChangeProtocol(void)
```

```
{
        switch (DloCfg.out_protocol) {
        case OUT_PPP:
                DPCTxFrame = DebugFlag ? PPPSendRoutineDebug : PPPSendRoutineOpt
;
                break;
        case OUT_NETWORK: {
                void (*ControlEntryPoint)(void) = 0;
                struct DriverConfigurationStructure* dvrCfg = 0;

                if (CLSLGetMLIDControlEntry(DloCfg.net_interface,
                                            &ControlEntryPoint))
                {
                        goto skipDriver;
                }
                dvrCfg = (struct DriverConfigurationStructure *)
                        CommandMlid(DloCfg.net_interface, 0, (LONG)ControlEntryP
oint);
                memcpy(RawEnvelope + 6, dvrCfg->DNodeAddress, 6);

        skipDriver:
                RawECB.ECB_BoardNumber = DloCfg.net_interface;
                memcpy(RawEnvelope, DloCfg.net_addr, 6);

                DPCTxFrame = DebugFlag ? RawSendRoutineDebug : RawSendRoutineOpt
;
                break;
        }
        case OUT_SLIP:
                DPCTxFrame = DebugFlag ? SLIPSendRoutineDebug : SLIPSendRoutineO
pt;
                break;
        }
        InetStateChange(DLOS_DISC_4);
        DloEndConn();
}


int ProcessLogin(void)
{
        BYTE value;
        char *sendStr, *waitStr;
        char sendBuf[40];
        LONG nextTimeout;

        /* No use trying if we aren't even connected */

        /* Get out if we're done */

        if (WaitingLines == 0 || DloCfg.auto_login == FALSE)
        {
                return(TRUE);
        }

        if (!DloConnected())
                return(FALSE);

        /* Timeout if we've waited too long for this wait */

        if (ConnectingTimeout == 0)
        {
                ConnectingTimeout = GetCurrentTime() + DloCfg.wait_timeout_1 * 1
8;
        }

        if (GetCurrentTime() > ConnectingTimeout)
        {
                if (ConnectingRedial == FALSE)
                {
                        /* First timeout. Send return and try again. */
                        ConnectingRedial = TRUE;
                        InitLogin();
                        DloSend(MSG("\r", 181), 1, DLO_INET_TIMEOUT);
                        return(FALSE);
                }
                DloEndConn();
                return(FALSE);
        }

        DisplayWaitStatus();

        while (DloReceive(&value, 1) != 0)
        {
                if (DebugFlag)
                        putchar(value);
                if (value != '\r' && value != '\n')
                {
                        WaitingBuffer[WaitingIndex++] = value;
                        WaitingBuffer[WaitingIndex] = 0;
                        if (WaitingIndex > (MAX_READ_BUF-1))
                                WaitingIndex = 0;
                }

                waitStr = (char *)&DloCfg.wait_for_1[NextWait * 30];
                if(strstr(WaitingBuffer, waitStr) != NULL)
                {
                        sendStr = (char *)&DloCfg.send_1[NextWait * 30];
                        NWsprintf(sendBuf, MSG("%s\r", 558), sendStr);
                        DloSend(sendBuf, CStrLen(sendBuf), DLO_INET_TIMEOUT);
                        NextWait++;
                        WaitingIndex = 0;
                        WaitingBuffer[WaitingIndex] = '\0';
                        WaitingLines--;
                        if (WaitingLines == 0)
                        {
                                DloUpdateModemStr();
                                return(TRUE);
                        }
                        DisplayWaitStatus();
                        nextTimeout = DloCfg.wait_timeout_1 + NextWait;
                        ConnectingTimeout = GetCurrentTime() +
                                        ((nextTimeout) ? (nextTimeout * 18) : (5
*18));
                        return(FALSE);
                } else if (value == '\r')
                {
                        WaitingIndex = 0;
                        WaitingBuffer[WaitingIndex] = '\0';
                }
        }

        return(FALSE);
}

int ConnectProtocol(void)
{
        int ccode;

        if (DloCfg.out_protocol == OUT_SLIP)
        {
                delay(1000);            /* time to "settle" */
                return 1;
```

```
        }
        /*PPP*/
        else
        {
                ccode = ConnectPPP();
        }
        return ccode;
}


void TinetProtocolBind(LONG __parameter) {
  struct EventProtocolBindStruct* epbs =
    (struct EventProtocolBindStruct*)__parameter;
  if (epbs->boardNumber == DIOBoard &&
      epbs->protocolNumber == 1/*PROTOCOL_ID_TCPIP*/) {
    extern LONG DPCNextRegistrationCheck;
    DPCGetIPAddress(&DPC_IP_Address);
    DPCNextRegistrationCheck = 0;
  }
}


/****************************************************************************
 *
 *      InetMain(void *parm)
 *
 *      Description:
 *              Main thread for Turbo Internet handling.
 *
 *      Input:
 *              parm                                            - ignored
 *
 *      Output:
 *              nothing
 *
 *      Returns:
 *              nothing
 *
 ****************************************************************************/

void InetMain(void *parm)
{
        time_t nextStartConn = 0;
        LONG removedCount = (LONG)(-1);
        long millidelay = 0;
        LONG protocolBindHandle =
          RegisterForEvent(EVENT_PROTOCOL_BIND, TinetProtocolBind, 0);

        parm = parm;                    /* unused */

        NewQ.semaphore = OpenLocalSemaphore(0);
        TxQ.semaphore = OpenLocalSemaphore(0);
        BeginThread(FilterQueue, 0, 0, 0);

        TxChainRTag = AllocateResourceTag(NLMHandle,
                                MSG("Turbo Inet TxPreScan Chain", 476)

                                LSLTxPreScanStackSignature);
        TxECBRTag = AllocateResourceTag(NLMHandle,
                                MSG("Turbo Inet Transmit Packets", 619),
                                ECBSignature);
        RxChainRTag = AllocateResourceTag(NLMHandle,
                                "Turbo Inet RxPreScan Chain",
                                LSLPreScanStackSignature);
        RxECBRTag = AllocateResourceTag(NLMHandle,
                                MSG("Turbo Inet Receive Packets", 619),
                                        ECBSignature);
        DPCGetIPAddress(&DPC_IP_Address);

        if (DloCfg.out_protocol == OUT_NETWORK)
          InetState = PROTOCOL_CONNECTED;

mainloop:
        while (!ExitingFlag)
        {
                InetAsleep = TRUE;
                if (millidelay > 55)
                  delay(millidelay);
                else if (millidelay > 0)
                  ThreadSwitchWithDelay/*LowPriority*/();
                else
                  ThreadSwitch();
                InetAsleep = FALSE;

                while (DIOBoard && removedCount != DIORemovedCount)
                {
                        BYTE address[8];
                        BYTE szBicBCDAddress[20];
                        struct DriverStatsStructure* stats = 0;
                        LONG ip_address = ntohl(DloCfg.ip_address);

                        removedCount = DIORemovedCount;
                        /* Enable internet reception */

                        /* Yuk. We'll change this later to get rid these extra s
tages */
                        NWsprintf(szBicBCDAddress, MSG("%d.%d.%d.%d", 620),
                                  (ip_address >> 24) & 0xff,
                                  (ip_address >> 16) & 0xff,
                                  (ip_address >> 8) & 0xff,
                                  (ip_address) & 0xff);

                        convert_address(szBicBCDAddress);
                        if (!pack_mac_addr(address, 6,
                                           szBicBCDAddress, CStrLen(szBicBCDAddr
ess)))
                        {
                                /* UpdateModemStr(MSG("ERROR: could not pack mac
 address      \n", xxxx)); */
                                millidelay = 500;
                                break;
                        }

                        /* Sending an esr address of -1 tells MLID to handle rec
eption */
                        if (DIOOpenChannel(address,
                                           (int (*)())0xffffffff,
                                           &InetChannel))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                        if (ExitingFlag)
                                break;
                        DIOAddHIAddr(InetChannel, (BYTE *)&HIAddr);
                        DPCGetMLIDStats(&stats);
                        DIOStats->TxOKMultipleCollisions = 0;
                        if (CLSLRegisterPreScanTxChain(TxChainRTag,
                                                        DIOBoard,
                                                        3, /* next to last */
                                                        &TxChainID,
```

```
                                                        InetQueuePacket,
                                                        InetControl,
                                                        TxECBRTag))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                        if (CLSLRegisterPreScanRxChain(RxChainRTag,
                                                        DIOBoard,
                                                        3, /* next to last */
                                                        &RxChainID,
                                                        ConnectionLimiter,
                                                        InetControl,
                                                        RxECBRTag))
                        {
                                millidelay = 500;
                                removedCount = (LONG)(-1);
                                break;
                        }
                }

                if (DloCfg.out_protocol == OUT_PPP &&
                    InetState == PROTOCOL_CONNECTED)
                {
                        PPPBackground();
                }

                if (TxQ.head == 0 &&
                    (InetState <= MODEM_CONNECTING ||
                     InetState >= PROTOCOL_CONNECTED))
                {
                  TimedWaitOnLocalSemaphore(TxQ.semaphore, 200);
                  millidelay = 0;
                  continue;
                }

                switch (InetState)
                {
                case MODEM_CONNECTED:
                        if (!ProcessLogin())
                        {
                                millidelay = 500;
                                break;
                        }
                        InetState = LOGIN_CONNECTED;
                        ;                       /* fallthru */
                case LOGIN_CONNECTED:
                        if (!ConnectProtocol())
                        {
                                DloEndConn();
                                millidelay = 15 * 1000;
                                break;
                        }
                        InetState = PROTOCOL_CONNECTED;
                        millidelay = 1;
                        break;
                case PROTOCOL_CONNECTED:
                {
                        LONG count = 0;
                        if (DloCfg.out_protocol != OUT_NETWORK &&
                            AIOWriteStatus(AIOPortHandle, &count, 0))
                        {
                                millidelay = 200;
                                break;
                        }
                        if (count == 0)
                        {
                                LONG milliclock = milliclock();
                                ECB* ecb;
                                ecb = TxQ.head;
                                millidelay = 100;
                                while (ecb->activityTimer > milliclock)
                                {
                                        LONG diff = ecb->activityTimer - millicl
ock;
                                        if (diff < millidelay)
                                                millidelay = diff;
                                        if ((ecb = ecb->ECB_NextLink) == 0)
                                                goto mainloop;
                                }
                                Remove(&TxQ, ecb);
                                if (ecb->activityTimer < milliclock() - 60000) {
                                  if (ecb->activityTimer)
                                    ++DIOStats->TxAbortExDeferral;
                                }
                                else
                                  IPSendRoutine(ecb);
                                ReleaseECB(ecb);
                                if (DloCfg.out_protocol != OUT_NETWORK)
                                  AIOWriteStatus(AIOPortHandle, &count, 0);
                        }
                        millidelay = (count *
                                       10 *      /* Tx bits with framing */
                                       1000 /    /* milliseconds */
                                       BaudRate[DloCfg.tinet_baud_index]);
                        break;
                }
                case MODEM_IDLE:
                        if (nextStartConn < time(0))
                        {
                                InitLogin();
                                DloStartConn(DLO_INET_TIMEOUT);
                                InetState = MODEM_CONNECTING;
                                nextStartConn = time(0) + 30;
                                ;               /* fallthru */
                        }
                default:
                        millidelay = 10 * 1000;
                        break;
                }
        }

        DIOCloseChannel(InetChannel);

        CLSLDeRegisterPreScanRxChain(RxChainID);
        CLSLDeRegisterPreScanTxChain(TxChainID);
        while (TxQ.head)
          ReleaseECB(Dequeue(&TxQ));
        while (NewQ.head)
          ReleaseECB(Dequeue(&NewQ));
        CloseLocalSemaphore(TxQ.semaphore); TxQ.semaphore = 0;
        CloseLocalSemaphore(NewQ.semaphore); NewQ.semaphore = 0;

        UnregisterForEvent(protocolBindHandle);

        DPCInetPID = 0;
        return;

}
```

UNITED STATES PATENT AND TRADEMARK OFFICE
DOCUMENT CLASSIFICATION BARCODE SHEET

As-Filed New Application

Level -1
Version 1.1